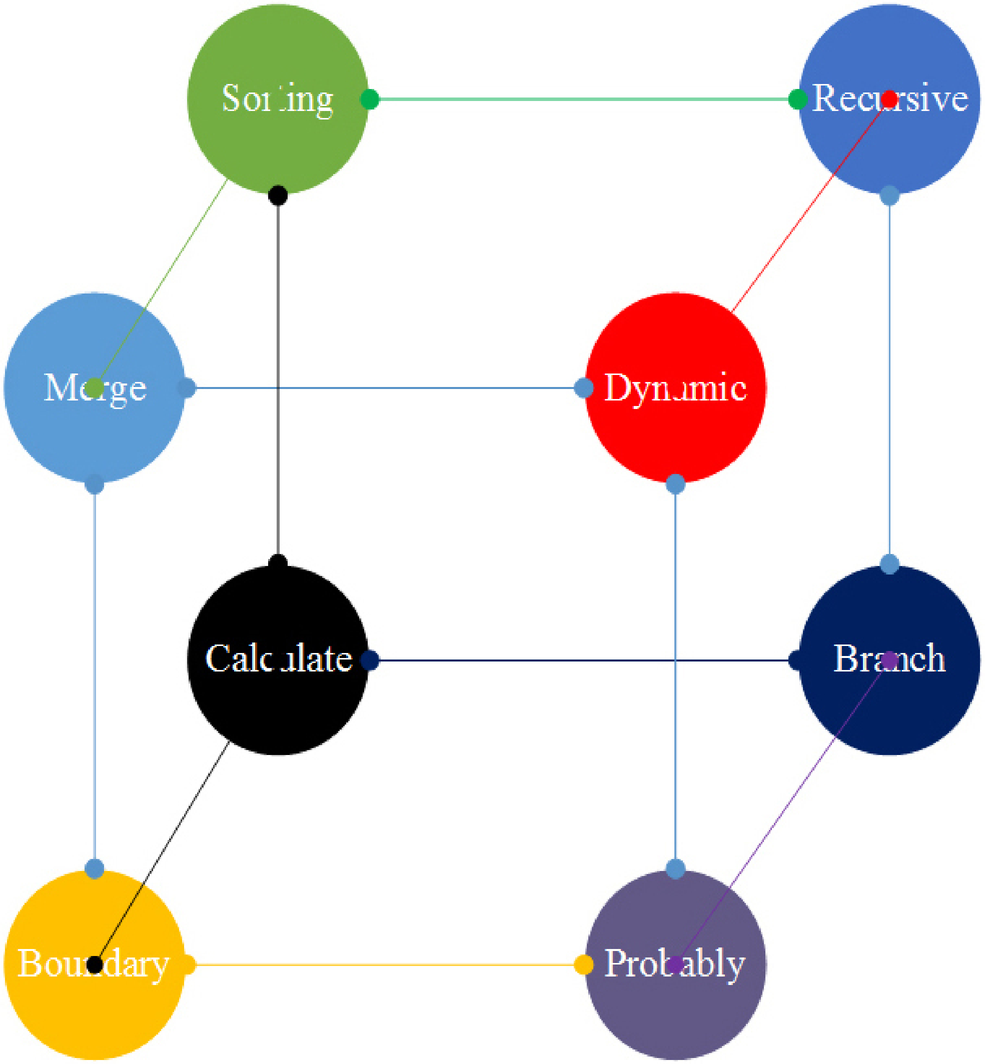
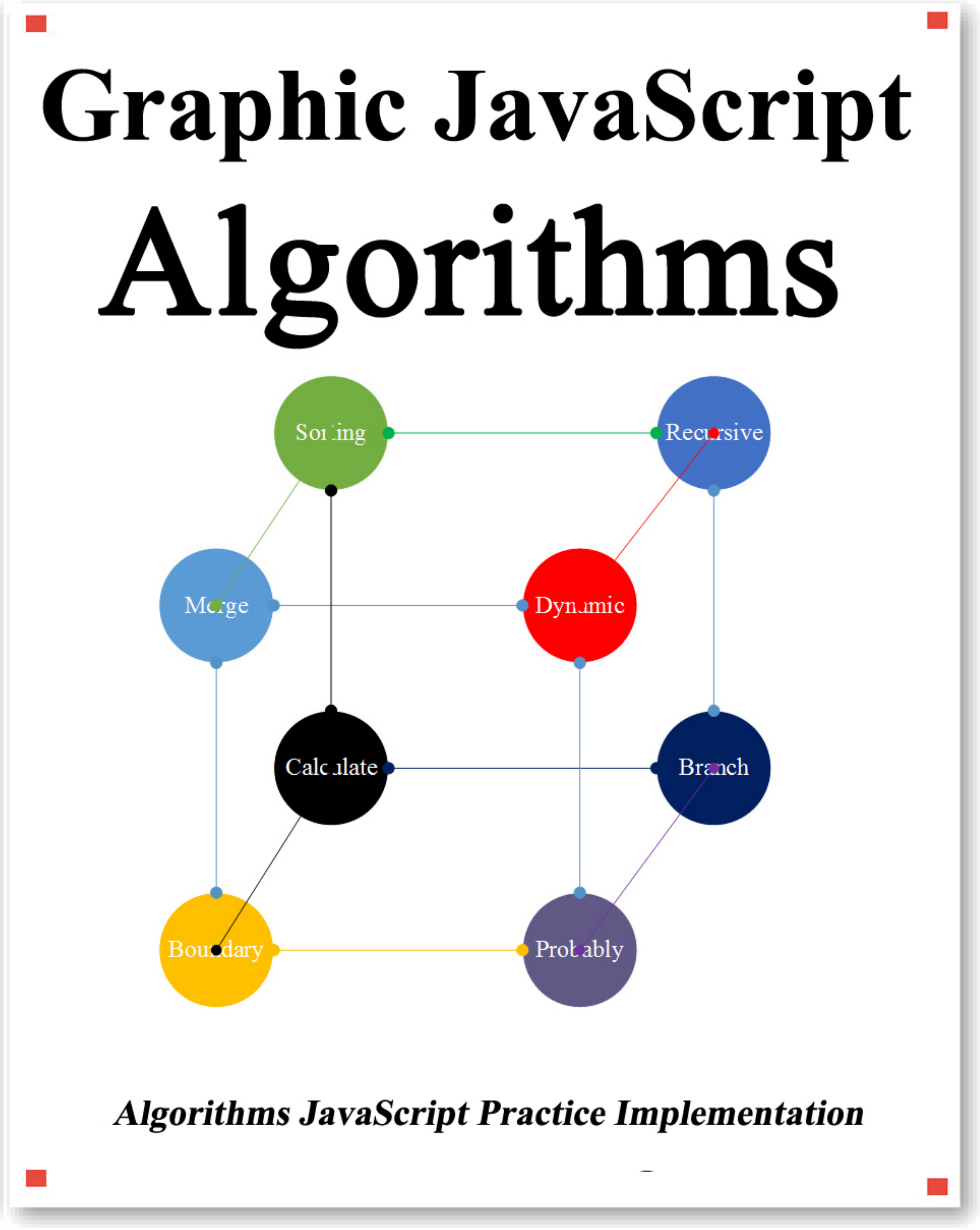
# Graphic JavaScript Algorithms


Sorting
Recursive
Merge
Dynamic
Calculate
Branch
Boundary
Probably

## *Algorithms JavaScript Practice Implementation*



# Graphic JavaScript Algorithms

Sorting
Recursive
Merge
Dynamic
Calculate
Branch
Boundary
Probably

***Algorithms JavaScript Practice Implementation***

# Graphic Javascript Algorithms

Sorting
Recursive
Merge
Dynamic
Calculate
Branch
Boundary
Probably

YANG HU

The complexity of life, because they do not understand to simplify the complex, simple is the beginning of wisdom. the essence of practice, to briefly explain the concept, and vividly cultivate programming interest, this book deeply analyzes Data Structures Algorithms ES6+ Javascript and fun of programming.

http://en.verejava.com

Copyright © 2020 Yang Hu

All rights reserved.

ISBN : 9798629790525

# CONTENTS

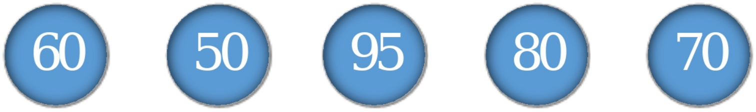
1 . Maximum Value
2. Bubble Sorting Algorithm
3 . Minimum Value

4. Select Sorting Algorithm
5. Insert Sorting Algorithm
6 . Reverse Array
7. Dichotomy Binary Search
8 . Shell Sorting
9. Unidirectional Linked List
    9.1 Create and Initialization
    9.2 Add Node
    9.3 Insert Node
    9.4 Delete Node
10. Doubly Linked List
    10.1 Create and Initialization
    10.2 Add Node
    10.3 Insert Node
    10.4 Delete Node
11. One-way Circular LinkedList
    11.1 Initialization and Traversal
    11.2 Insert Node
    11.3 Delete Node
12. Two-way Circular LinkedList
    12.1 Initialization and Traversal
    12.2 Insert Node
    12.3 Delete Node
13. Queue
14. Stack

15. Recursive Algorithm
16. Two-way Merge Algorithm
17. Quick Sort Algorithm
18. Binary Search Tree
    18.1 Construct a binary search tree
    18.2 Binary search tree In-order traversal
    18.3 Binary search tree Pre-order traversal
    18.4 Binary search tree Post-order traversal
    18.5 Binary search tree Maximum and minimum
    18.6 Binary search tree Delete Node
19. Binary Heap Sorting
20. Hash Table
21. Graph
    21.1 Undirected Graph and Depth-Frst Search
    21.2 Undirected Graph and Breadth-First Search
    21.3 Directed Graph and Depth-Frst Search
    21.4 Directed Graph and Breadth-First Search
    21.5 Directed Graph Topological Sorting
22 . Towers of Hanoi
23 . Fibonacci
24 . Dijkstra
25 . Mouse Walking Maze
26 . Eight Coins
27 . Josephus Problem

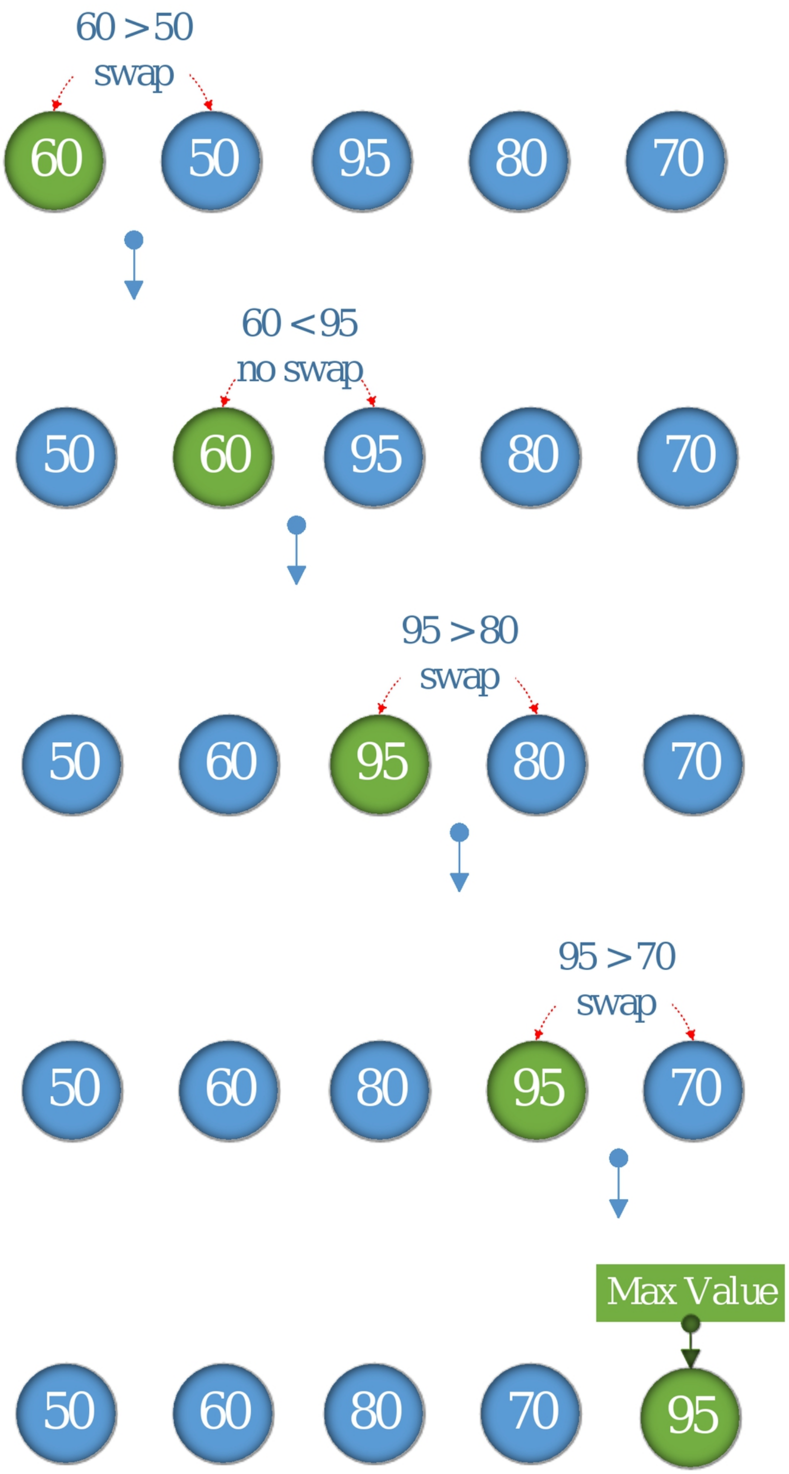
# Maximum Value

## Computing the Maximum of Integer Sequences:



### 1 . Algorithmic ideas

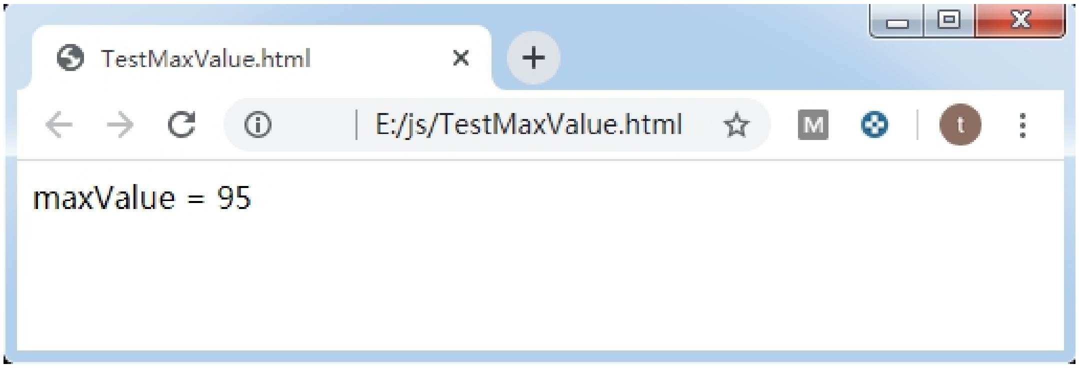
Compare arrays[i] with arrays[i + 1] , if arrays[i] > arrays[i + 1] are exchanged. So continue until the last number, arrays[length - 1] is the maximum.


60 > 50
swap
60 50 95 80 70
60 < 95
no swap
50 60 95 80 70
95 > 80
swap
50 60 95 80 70
95 > 70
swap
50 60 80 95 70
Max Value
50 60 80 70 95

## 1. Create a TestMaxValue.html with Notepad and open it in your browser.

```
<script type ="text/javascript" >
  function max( arrays) {
    // Maximum initialization value is 0
    for ( var i = 0 ; i < arrays.length - 1 ; i++) {
      if ( arrays[ i] > arrays[ i + 1 ]) { // swap
        var temp = arrays[ i];
        arrays[ i] = arrays[ i + 1 ];
        arrays[ i + 1 ] = temp;
      }
    }
    var maxValue = arrays[ arrays.length - 1 ];
    return maxValue;
  }

  //////////////////////testing////////////////////

  var scores = [ 60 , 50 , 95 , 80 , 70 ];
  var maxValue = max( scores);
  document.write( "maxValue = " + maxValue);

</script>
```

**Result:**

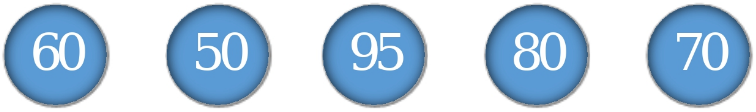


```
maxValue = 95
```

# Bubble Sorting Algorithm

**Bubble Sorting Algorithm:**
Compare arrays[j] with arrays[j + 1] , if arrays[j] > arrays[j + 1] are exchanged.
Remaining elements repeat this process, until sorting is completed.

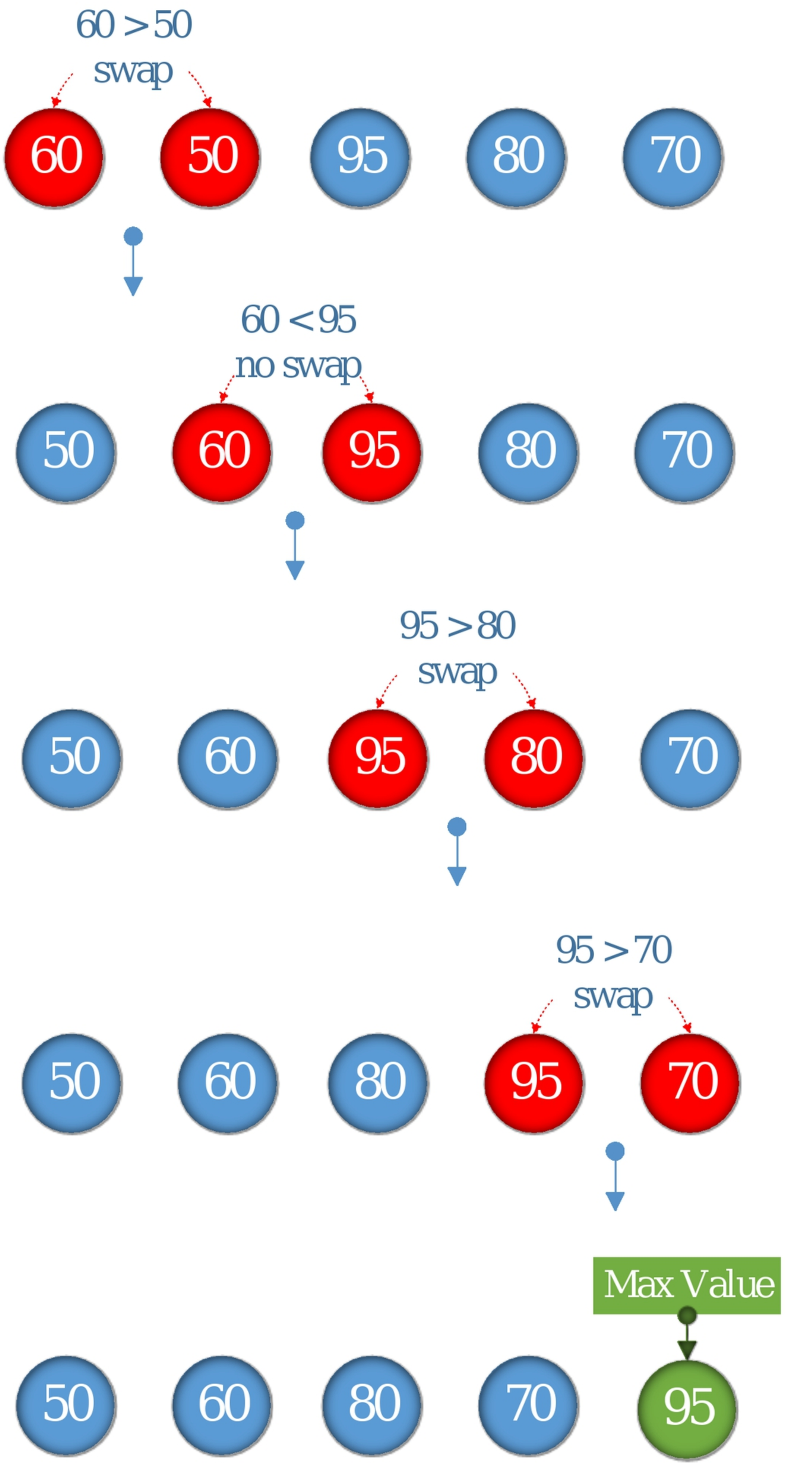
**Sort the following numbers from small to large**



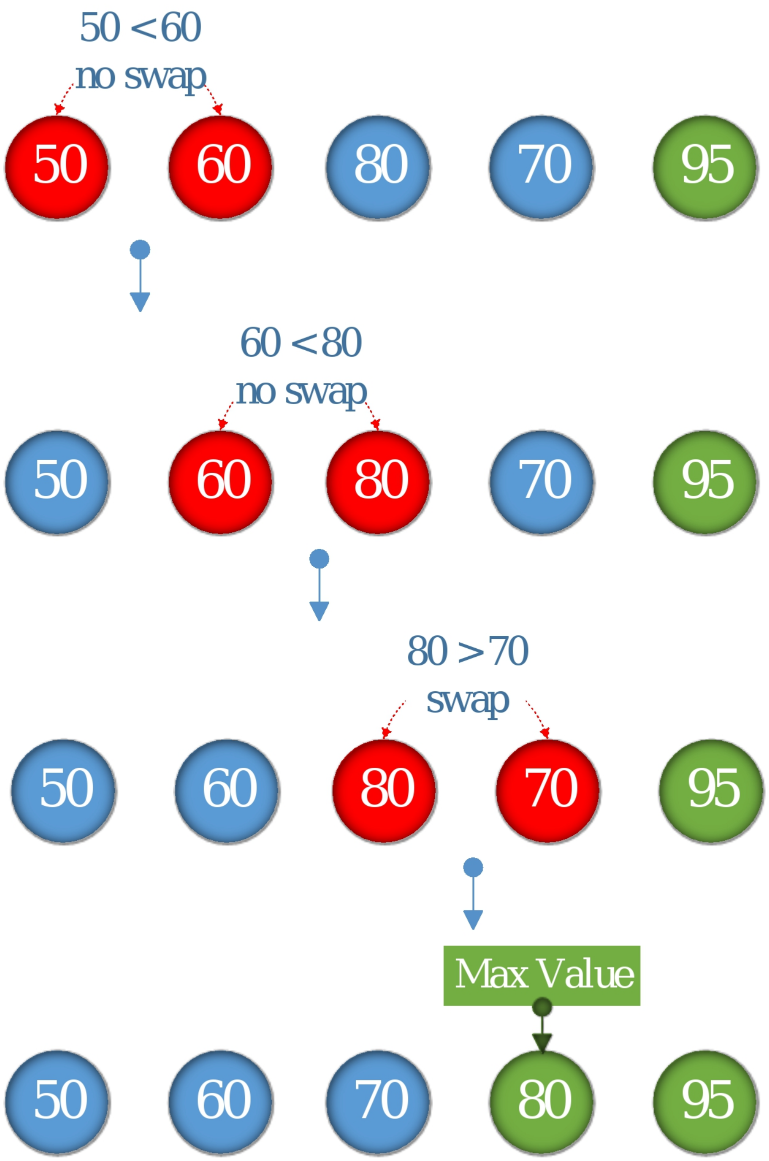
**Explanation:**

🔵 No sorting, 🔴 Comparing, 🟢 Already sorted

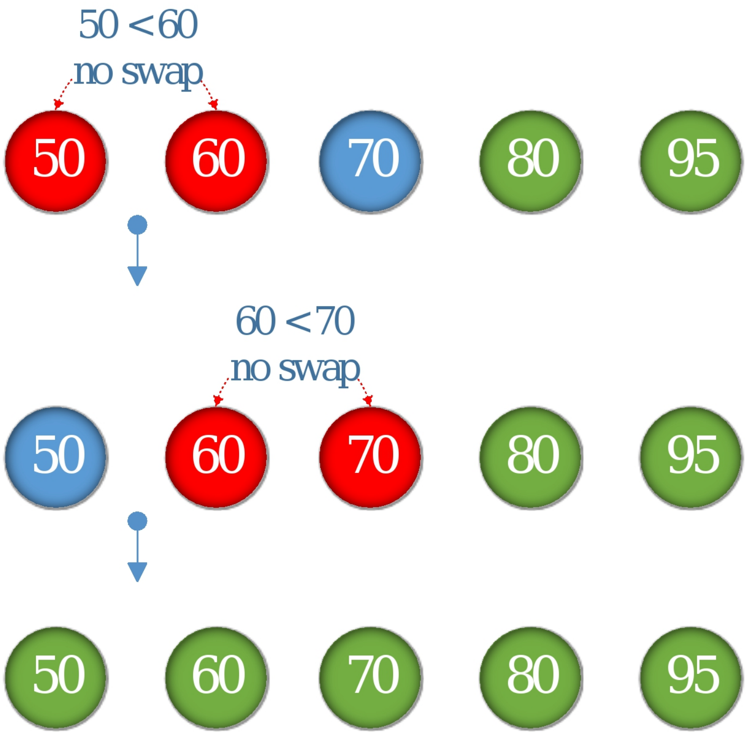
## 1. First sorting:


60 > 50
swap
60
50
95
80
70
60 < 95
no swap
50
60
95
80
70
95 > 80
swap
50
60
95
80
70
95 > 70
swap
50
60
80
95
70
Max Value
50
60
80
70
95

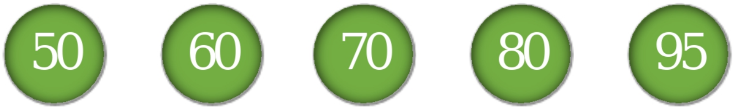
## 2. Second sorting:


50 < 60
no swap
60 < 80
no swap
80 > 70
swap
Max Value

**3． Third sorting:**


50 <60
no swap
50
60
70
80
95
60 <70
no swap
50
60
70
80
95
50
60
70
80
95

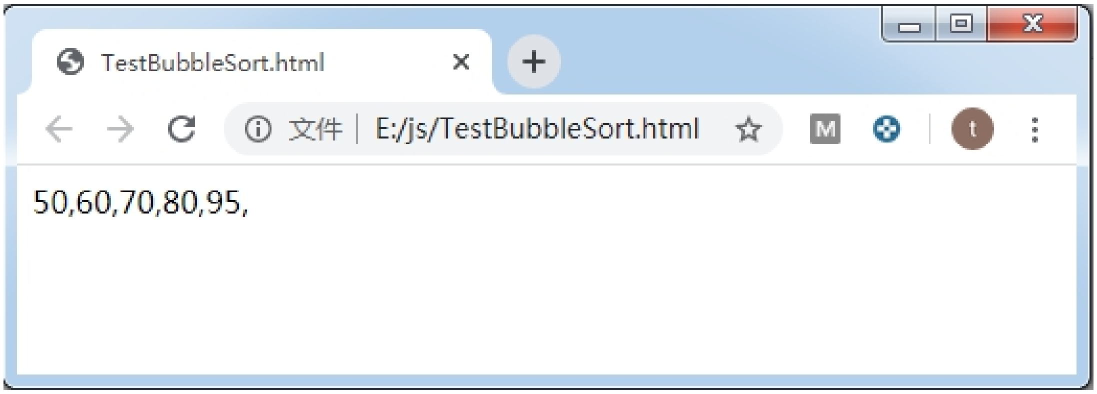
**No swap so terminate sorting : we can get the sorting numbers from small to large**


50
60
70
80
95

**1. Create a TestBubbleSort.html with Notepad and open it in your browser.**

```
<script type ="text/javascript" >
   class BubbleSort{
      static sort( arrays) {
         for ( var i = 0 ; i < arrays.length - 1 ; i++) {
            for ( var j = 0 ; j < arrays.length - i - 1 ; j++) {
               //swap
               if ( arrays[ j] > arrays[ j + 1 ]) {
                  var flag = arrays[ j];
                  arrays[ j] = arrays[ j + 1 ];
                  arrays[ j + 1 ] = flag;
               }
            }
         }
      }
   }

   //////////////////////testing////////////////////

   var scores = [ 60 , 50 , 95 , 80 , 70 ];
   BubbleSort.sort( scores);
   for ( var i = 0 ; i < scores.length; i++) {
      document.write( scores[ i] + "," );
   }
</script>
```

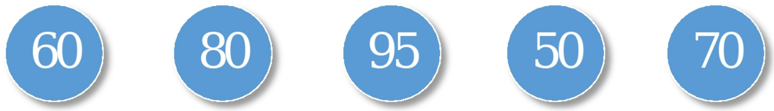
**Result:**


TestBubbleSort.html
文件
E:/js/TestBubbleSort.html

```
50,60,70,80,95,
```

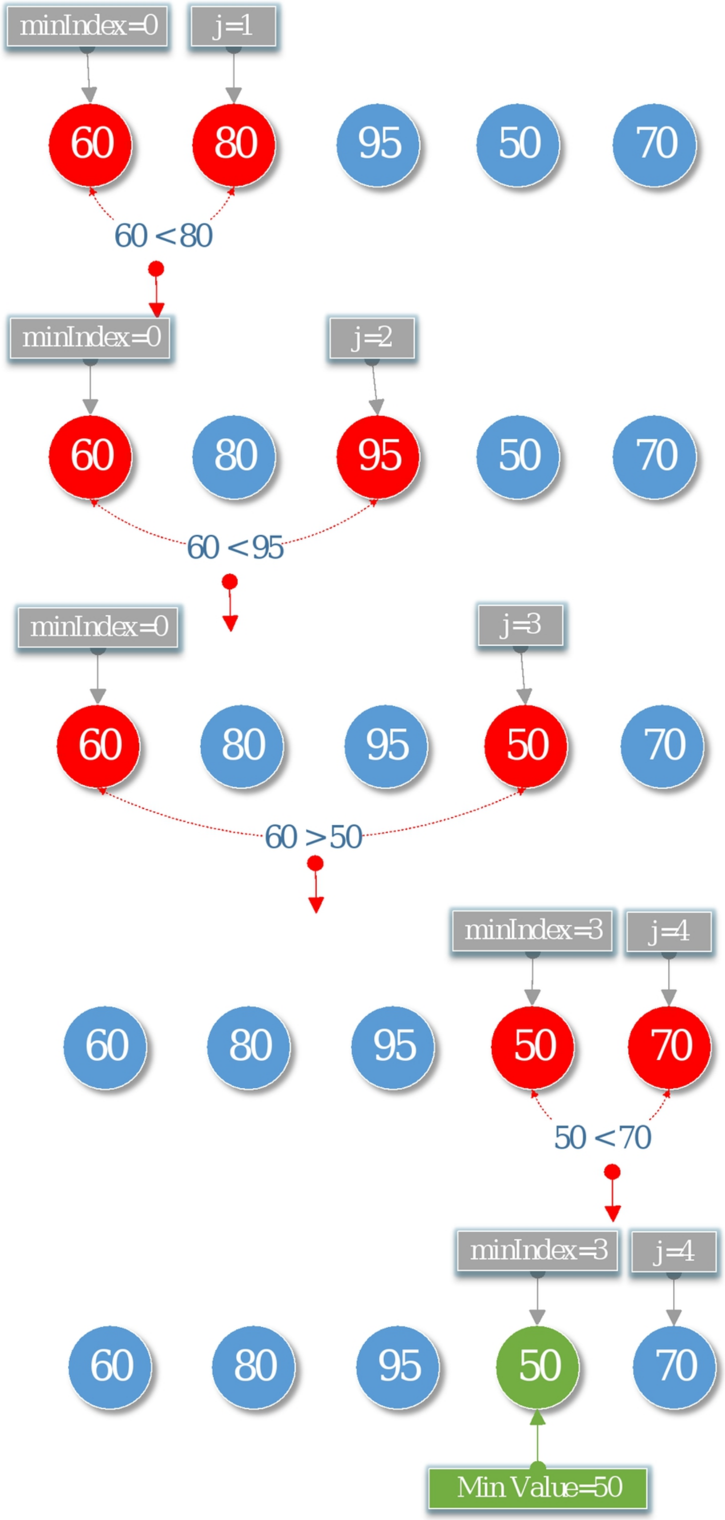
# Minimum Value

**Search the Minimum of Integer Sequences:**



### 1． Algorithmic ideas

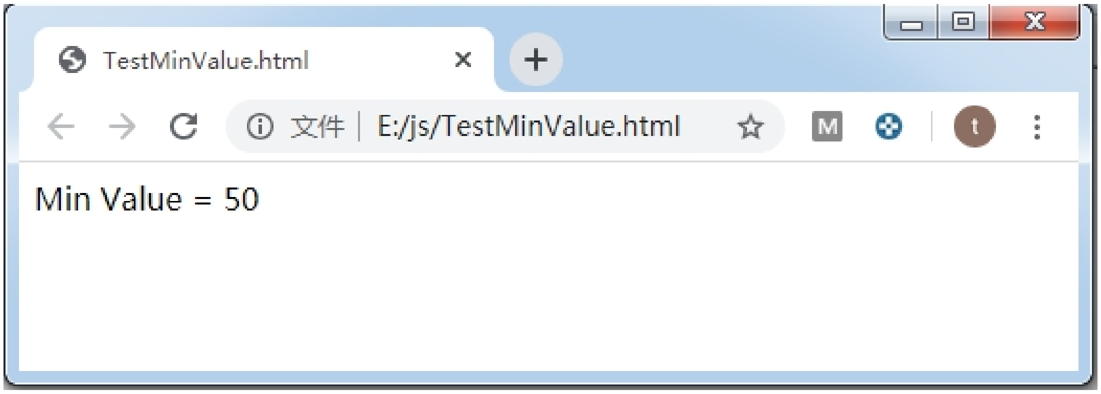
Initial value minIndex=0, j=1 Compare arrays[minIndex] with arrays[j] if arrays[minIndex] > arrays[j] then minIndex=j, j++ else j++. continue until the last number, arrays[minIndex] is the Min Value.


minIndex=0
j=1
60 < 80
minIndex=0
j=2
60 < 95
minIndex=0
j=3
60 > 50
minIndex=3
j=4
50 < 70
minIndex=3
j=4
Min Value=50

**1. Create a TestMinValue.html with Notepad and open it in your browser.**

```
<script type ="text/javascript" >
  function min( arrays) {
    var minIndex = 0 ; // the index of the minimum
    for ( var j = 1 ; j < arrays.length; j++) {
      if ( arrays[ minIndex] > arrays[ j]) {
        minIndex = j;
      }
    }
    return arrays[ minIndex];
  }

  //////////////////////testing////////////////////

  var scores = [ 60 , 80 , 95 , 50 , 70 ];
  var minValue = min( scores);
  document.write( "Min Value = " + minValue);

</script>
```

**Result:**

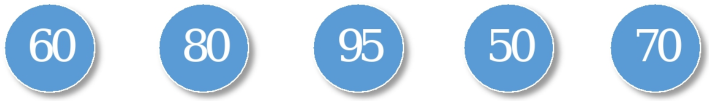

TestMinValue.html
文件 | E:/js/TestMinValue.html

```
Min Value = 50
```

# Select Sorting Algorithm

**Select Sorting Algorithm:**
Select the smallest one of the remaining unsorted elements in the array, inserting the one that has been sorted before.

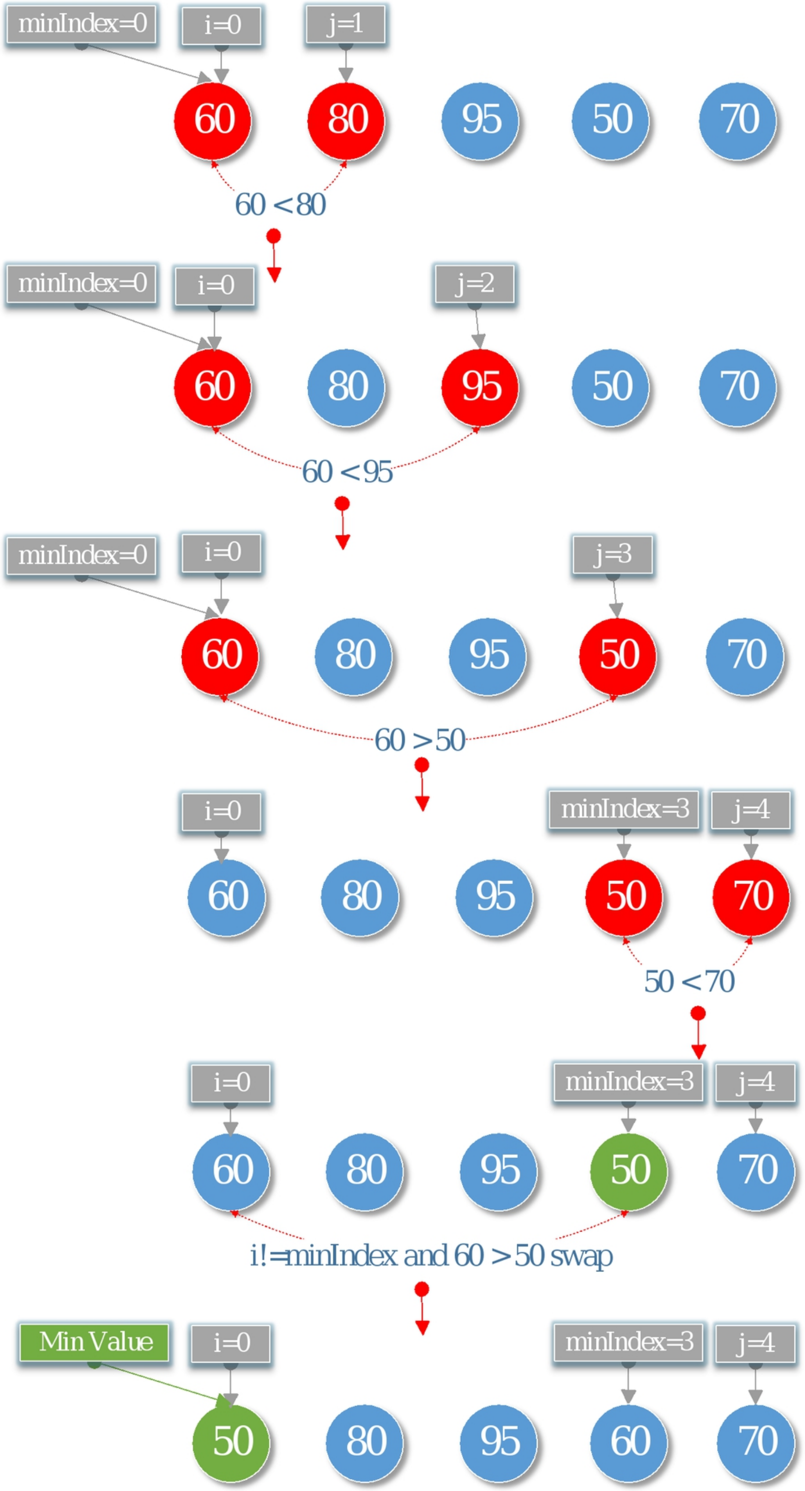
**Sort the following numbers from small to large**



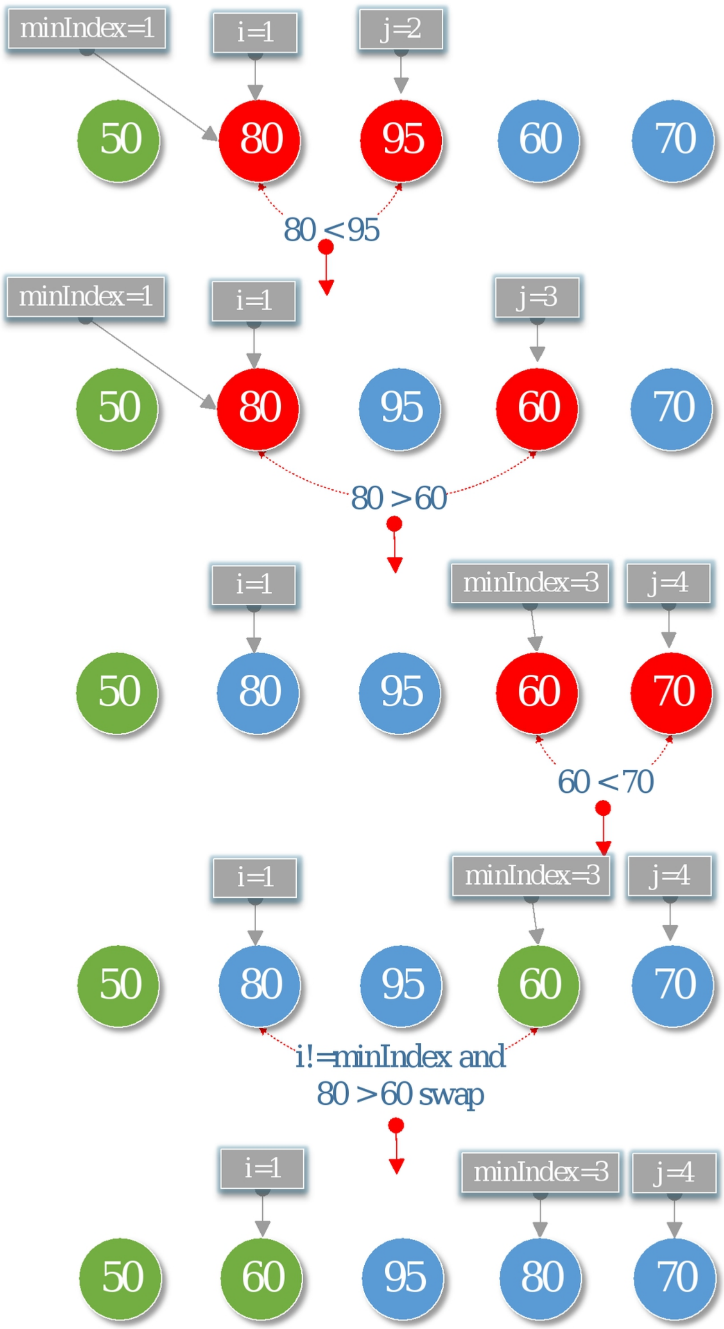
**Explanation:**

🔵 No sorting, 🔴 Comparing, 🟢 Already sorted.

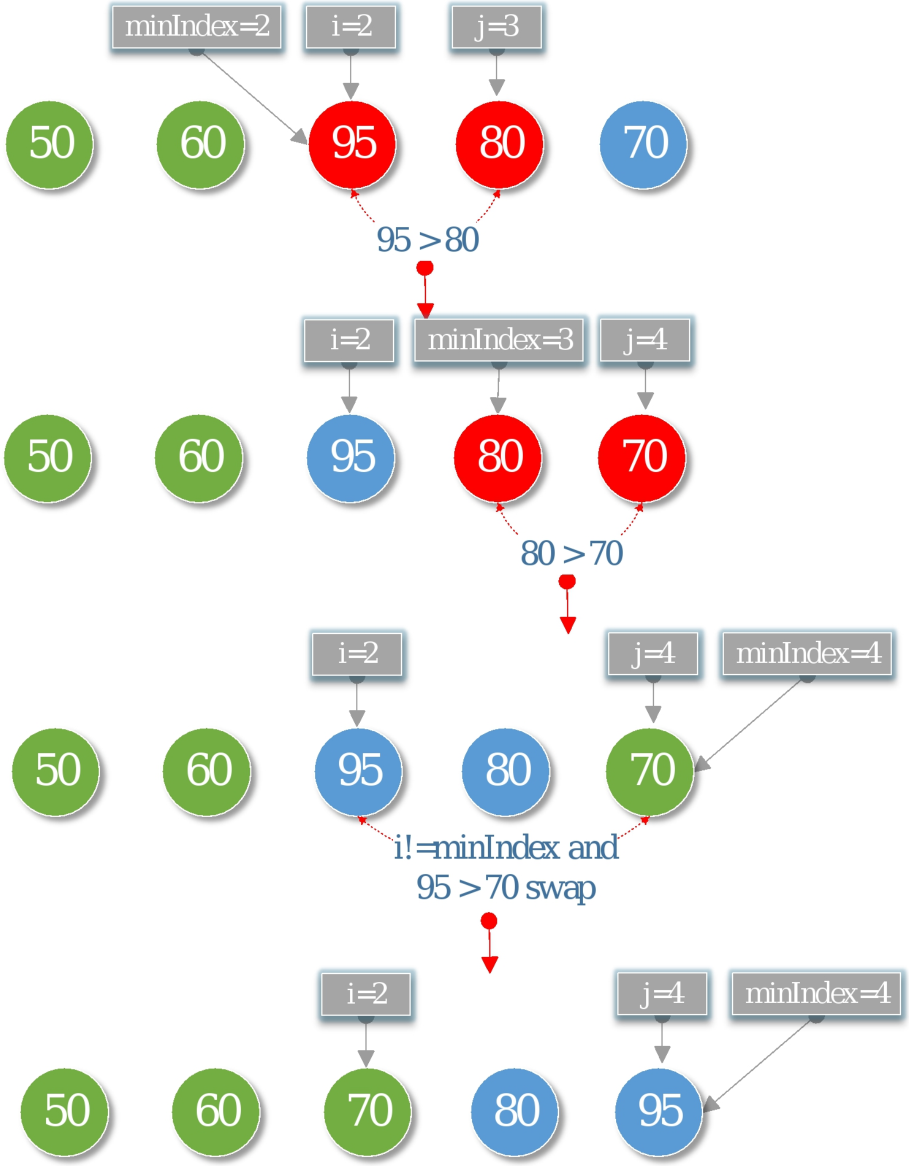
## 1 . First sorting:


minIndex=0
i=0
j=1
60
80
95
50
70
60 <80
minIndex=0
i=0
j=2
60
80
95
50
70
60 <95
minIndex=0
i=0
j=3
60
80
95
50
70
60 >50
i=0
minIndex=3
j=4
60
80
95
50
70
50 <70
i=0
minIndex=3
j=4
60
80
95
50
70
i!=minIndex and 60 >50 swap
Min Value
i=0
minIndex=3
j=4
50
80
95
60
70

## 2 . Second sorting:

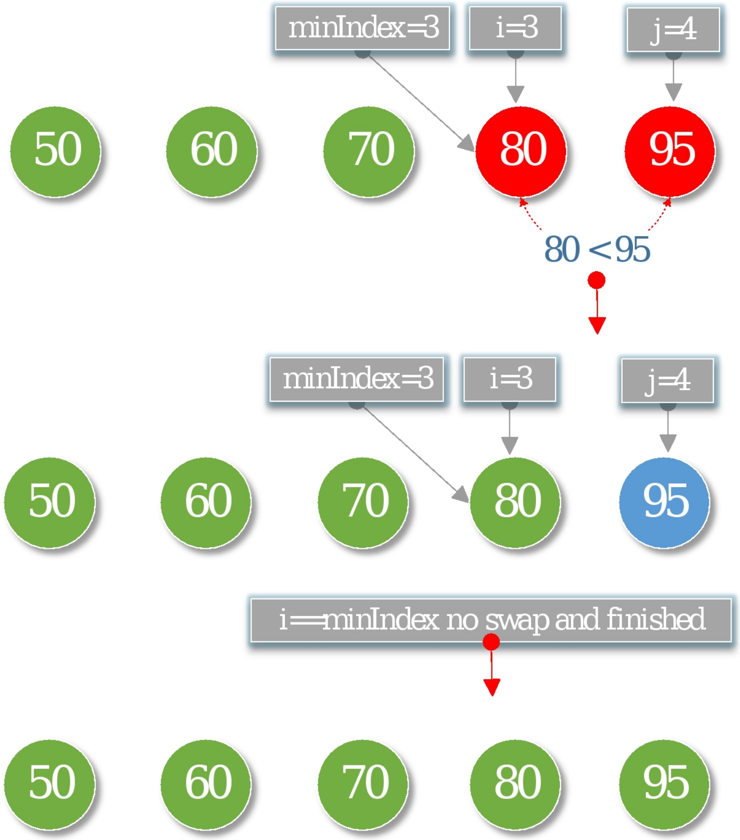

minIndex=1
i=1
j=2
50
80
95
60
70
80 < 95
minIndex=1
i=1
j=3
50
80
95
60
70
80 > 60
i=1
minIndex=3
j=4
50
80
95
60
70
60 < 70
i=1
minIndex=3
j=4
50
80
95
60
70
i!=minIndex and
80 > 60 swap
i=1
minIndex=3
j=4
50
60
95
80
70

## 3 . Third sorting:



minIndex=2
i=2
j=3
50
60
95
80
70
95 > 80
i=2
minIndex=3
j=4
50
60
95
80
70
80 > 70
i=2
j=4
minIndex=4
50
60
95
80
70
i!=minIndex and
95 > 70 swap
i=2
j=4
minIndex=4
50
60
70
80
95

### 4 . Forth sorting:


minIndex=3
i=3
j=4
50
60
70
80
95
80 <95
minIndex=3
i=3
j=4
50
60
70
80
95
i==minIndex no swap and finished
50
60
70
80
95

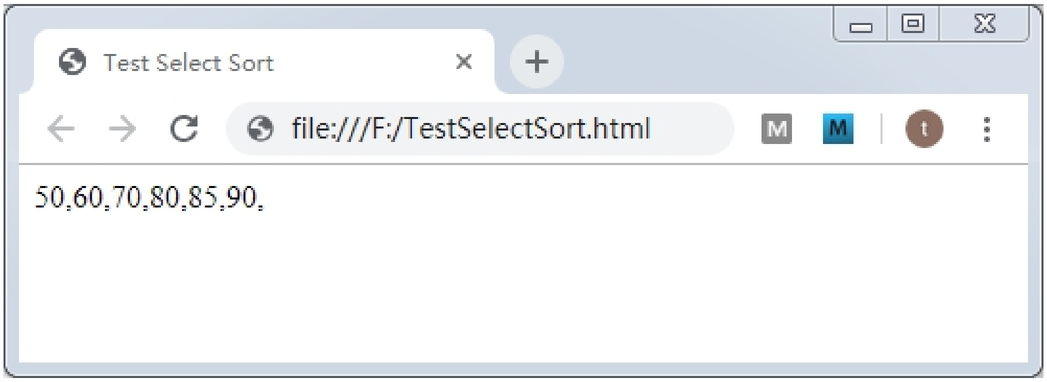
**we can get the sorting numbers from small to large**

50
60
70
80
95

**1. Create a TestSelectSort.html with Notepad and open it in your browser.**

```
<script type ="text/javascript" >
    class SelectSort{
        static sort( arrays) {
            var len = arrays.length - 1 ;
            var minIndex; // Save the index of the selected minimum

            for ( var i = 0 ; i < len; i++) {
                minIndex = i;
                //Save the minimum value of each loop as the first element
                var minValue = arrays[ minIndex];
                for ( var j = i; j < len; j++) {
                    // Compare with each element if it is less than the minimum value, exchange the minIndex
                    if ( minValue > arrays[ j + 1 ]) {
                        minValue = arrays[ j + 1 ];
                        minIndex = j + 1 ;
                    }
                }

                //if the minimum index changes, the current minimum is exchanged with the minIndex
                if ( i != minIndex){
                    var temp = arrays[ i];
                    arrays[ i] = arrays[ minIndex];
                    arrays[ minIndex] = temp;
                }
            }
        }
    }

    ////////////////////testing//////////////////

    var scores = [ 90 , 70 , 50 , 80 , 60 , 85 ];
    SelectSort.sort( scores);

    for ( var i = 0 ; i < scores.length; i++) {
        document.write( scores[ i] + "," );
    }
</script>
```

## Result:



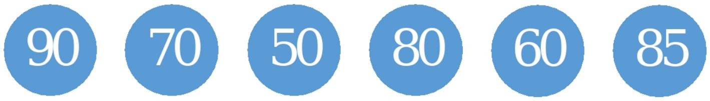
Test Select Sort
file:///F:/TestSelectSort.html

```
50,60,70,80,85,90,
```

# Insert Sorting Algorithm

**Insert Sorting Algorithm:**
Take an unsorted new element in the array, compare it with the already sorted element before, insert a new element to the right if the new element is smaller than the sorted element.

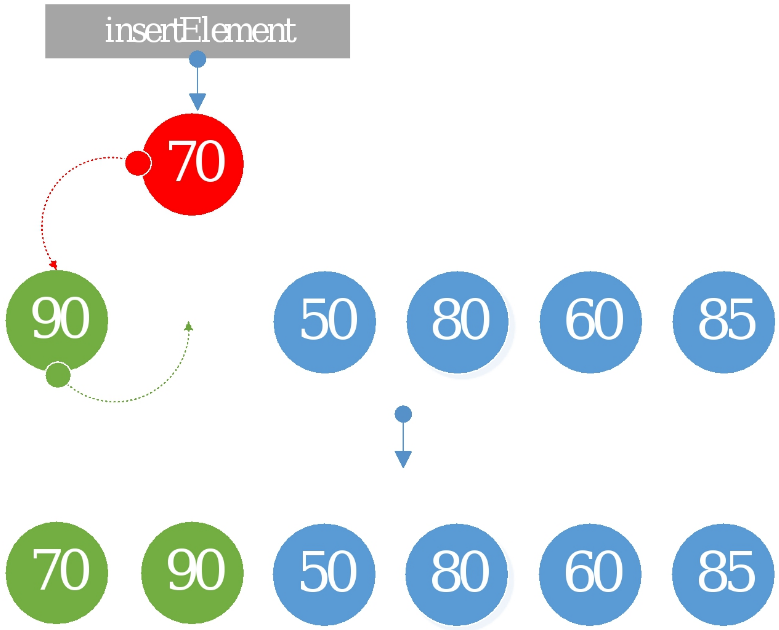
**Sort the following numbers from small to large**


90
70
50
80
60
85

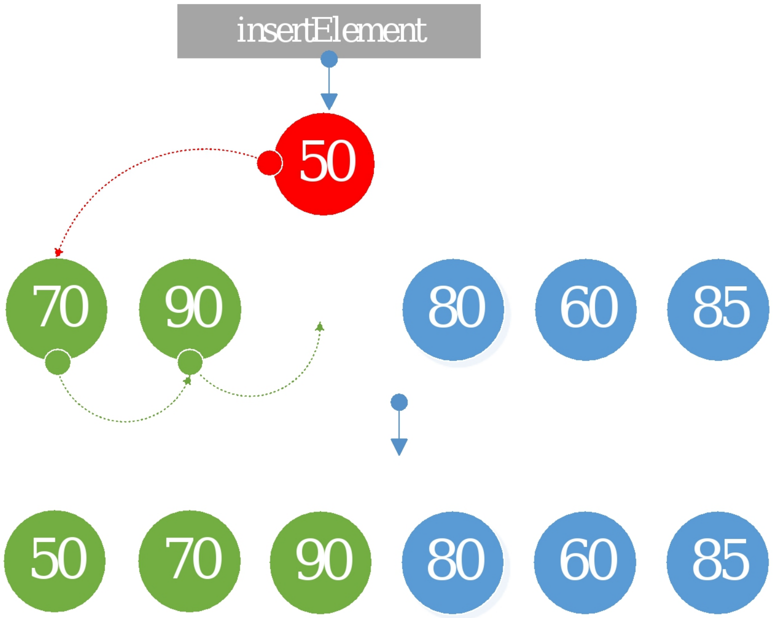
**Explanation:**

🔵 no sorting, 🔴 inserting, 🟢 Already ordered

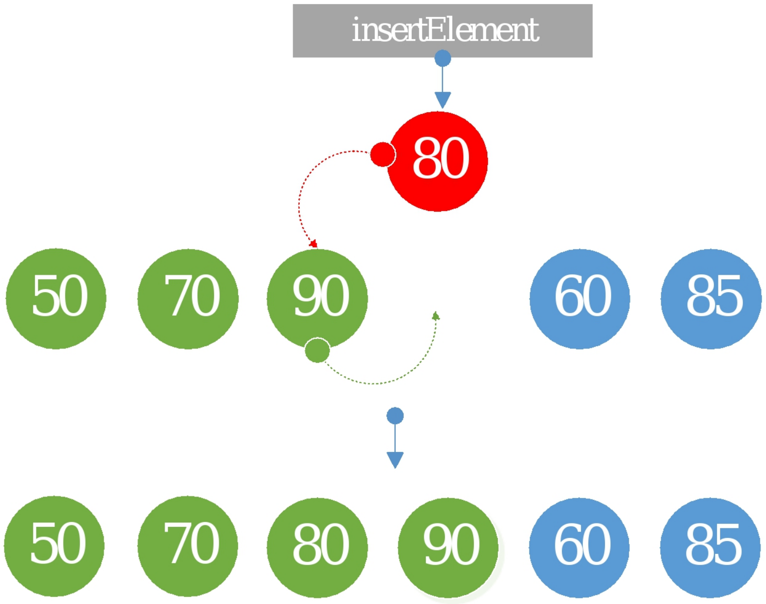
**1. First sorting:**


insertElement
70
90
50
80
60
85
70
90
50
80
60
85

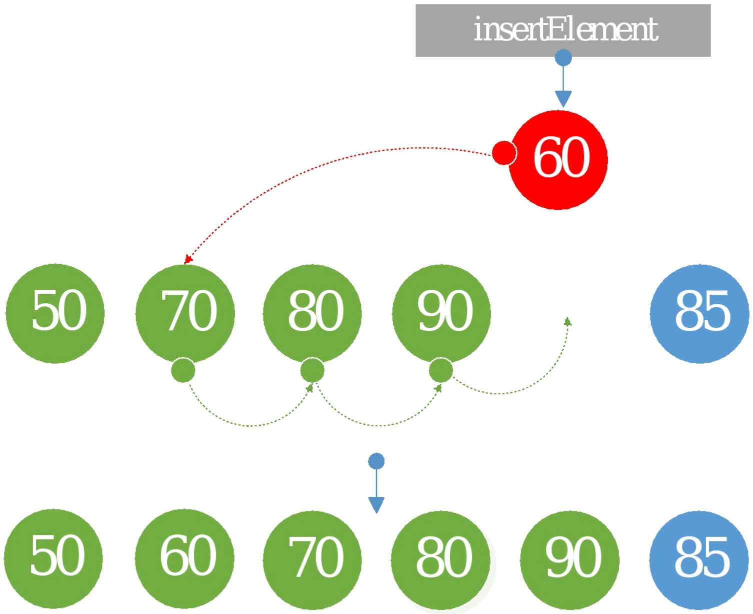
## 2 . Second sorting:


insertElement
50
70
90
80
60
85
50
70
90
80
60
85

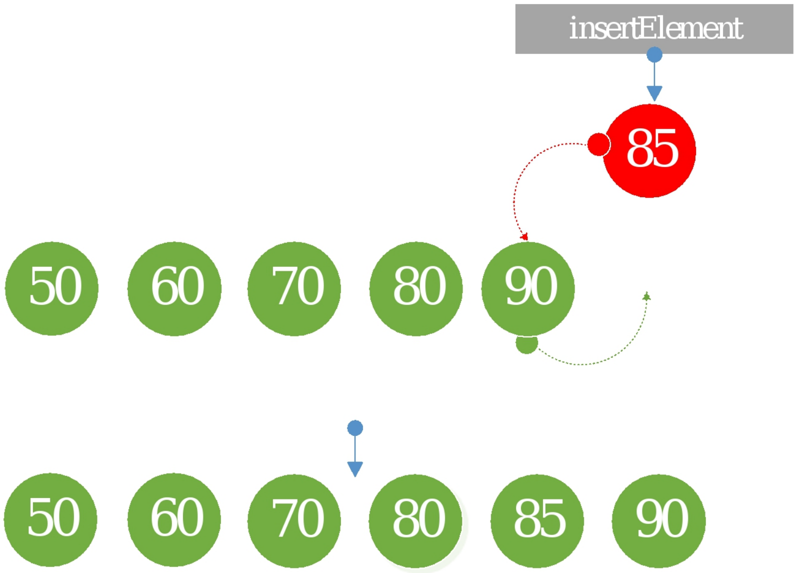
## 3 . Third sorting:


insertElement
80
50
70
90
60
85
50
70
80
90
60
85

## 4 Third sorting:


insertElement
60
50
70
80
90
85
50
60
70
80
90
85

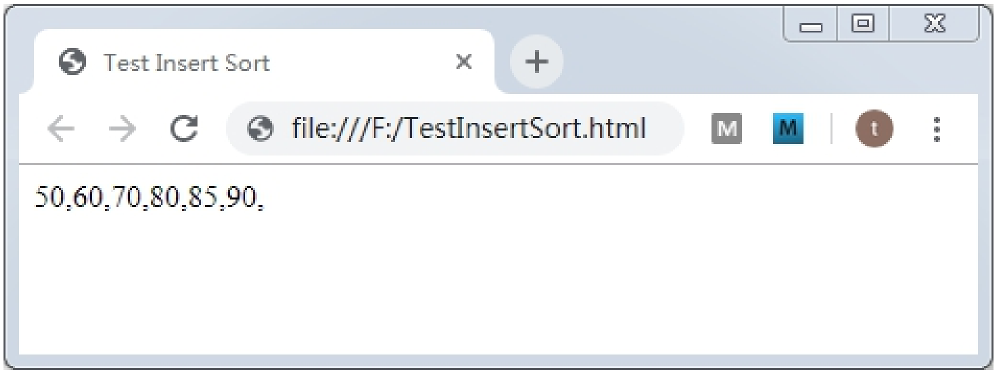
## 5 Fifth sorting:


insertElement
85
50
60
70
80
90
50
60
70
80
85
90

**1. Create a TestInsertSort.html with Notepad and open it in your browser.**

```
<script type ="text/javascript" >
    class InsertSort{
        static sort( arrays) {
            for ( var i = 0 ; i < arrays.length; i++) {
                var insertElement = arrays[ i]; //Take unsorted new elements
                var insertPosition = i; //Inserted position
                for ( var j = insertPosition - 1 ; j >= 0 ; j--) {
                    //If the new element is smaller than the sorted element, shifted to the right
                    if ( insertElement < arrays[ j]) {
                        arrays[ j + 1 ] = arrays[ j];
                        insertPosition--;
                    }
                }
                arrays[ insertPosition] = insertElement; //Insert the new element
            }
        }
    }

    //////////////////////testing////////////////////

    var scores = [ 90 , 70 , 50 , 80 , 60 , 85 ];
    InsertSort.sort( scores);
    for ( var i = 0 ; i < scores.length; i++) {
        document.write( scores[ i] + "," );
    }
</script>
```

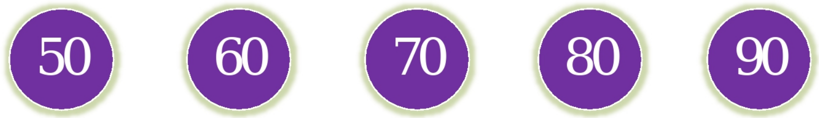
**Result:**


Test Insert Sort
file:///F:/TestInsertSort.html

```
50,60,70,80,85,90,
```

# Reverse Array

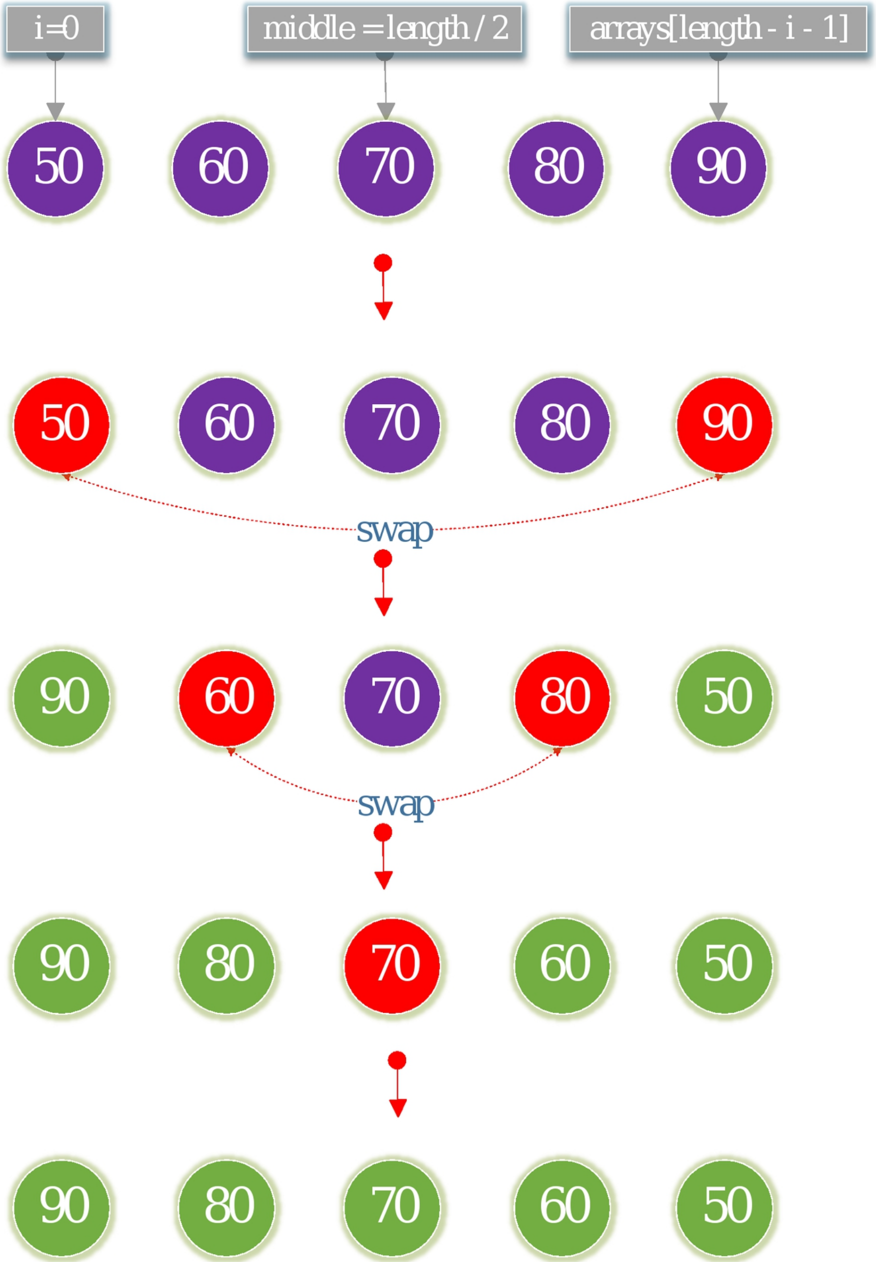
**Inversion of ordered sequences:**



## 1． Algorithmic ideas

Initial i = 0 and then swap the first element arrays[i] with last element arrays[length - i - 1]

Repeat until index of middle i == length / 2 .

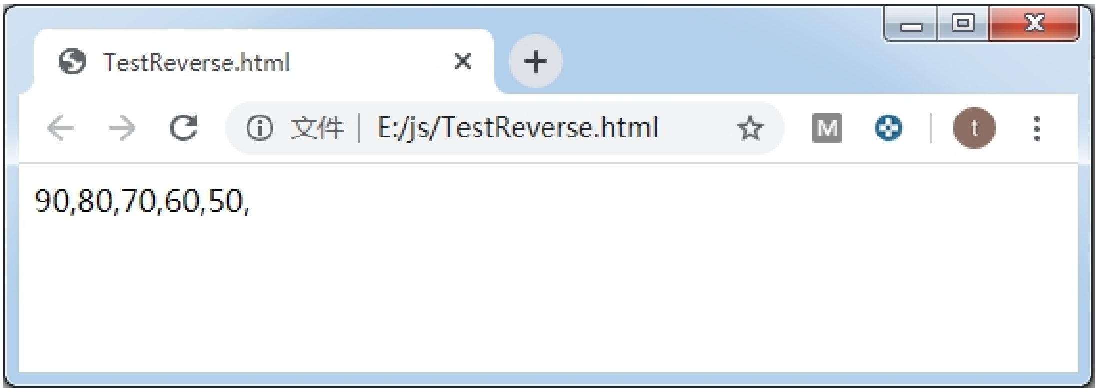


i=0
middle =length / 2
arrays[length - i - 1]
swap
swap

**1. Create a TestReverse.html with Notepad and open it in your browser.**

```
<script type ="text/javascript" >
  function reverse( arrays) {
      var length = arrays.length;
      var middle = length / 2 ;
      for ( var i = 0 ; i <= middle; i++) {
          var temp = arrays[ i];
          arrays[ i] = arrays[ length - i - 1 ];
          arrays[ length - i - 1 ] = temp;
      }
  }

  //////////////////////testing////////////////////

  var scores = [ 50 , 60 , 70 , 80 , 90 ];
  reverse( scores);
  for ( var i = 0 ; i < scores.length; i++) {
      document.write( scores[ i] + "," );
  }

</script>
```

**Result:**


TestReverse.html
文件 | E:/js/TestReverse.html
t

```
90,80,70,60,50,
```

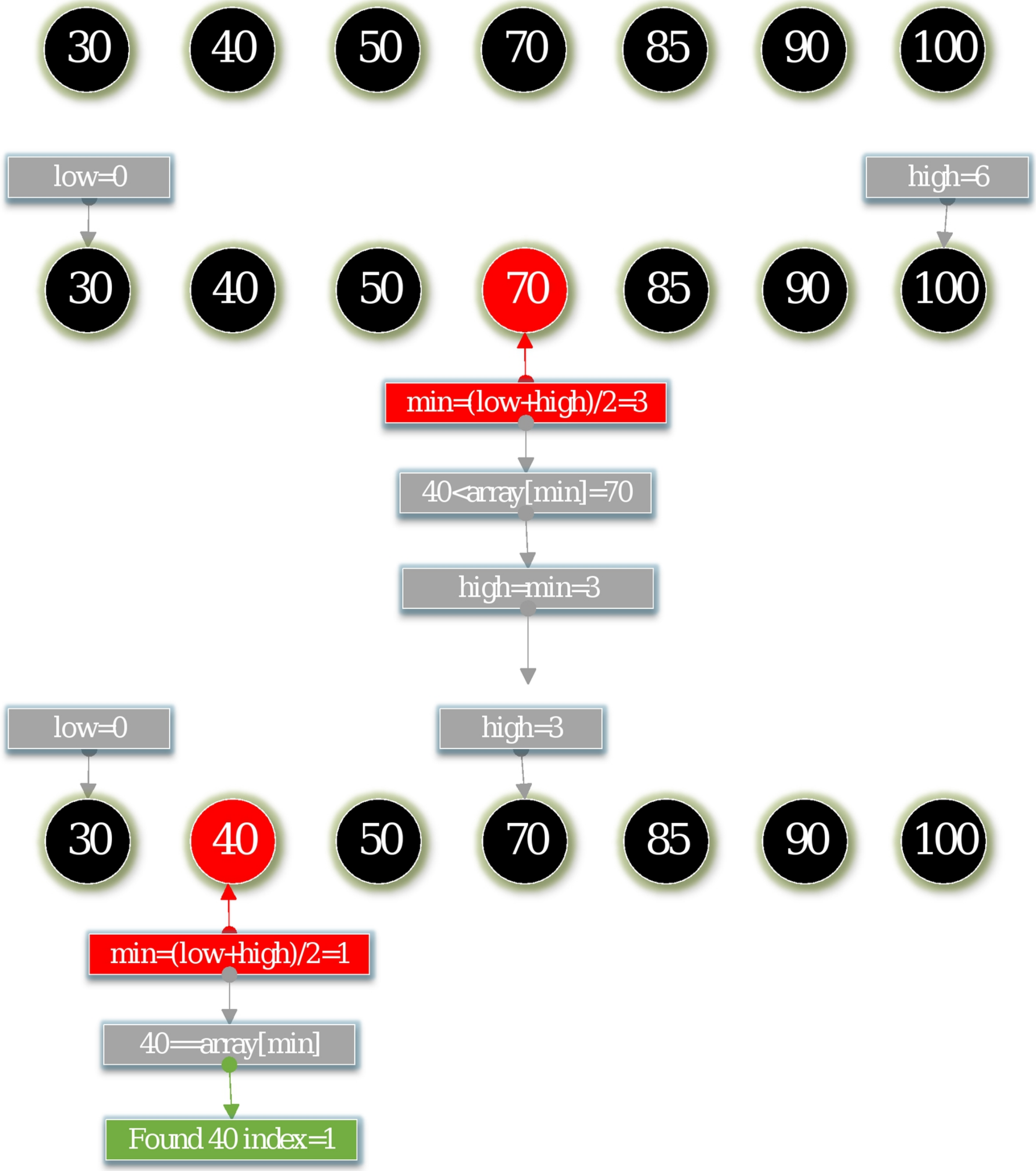
# Dichotomy Binary Search

**Dichotomy Binary Search:**
Find the index position of a given value from an already ordered array.

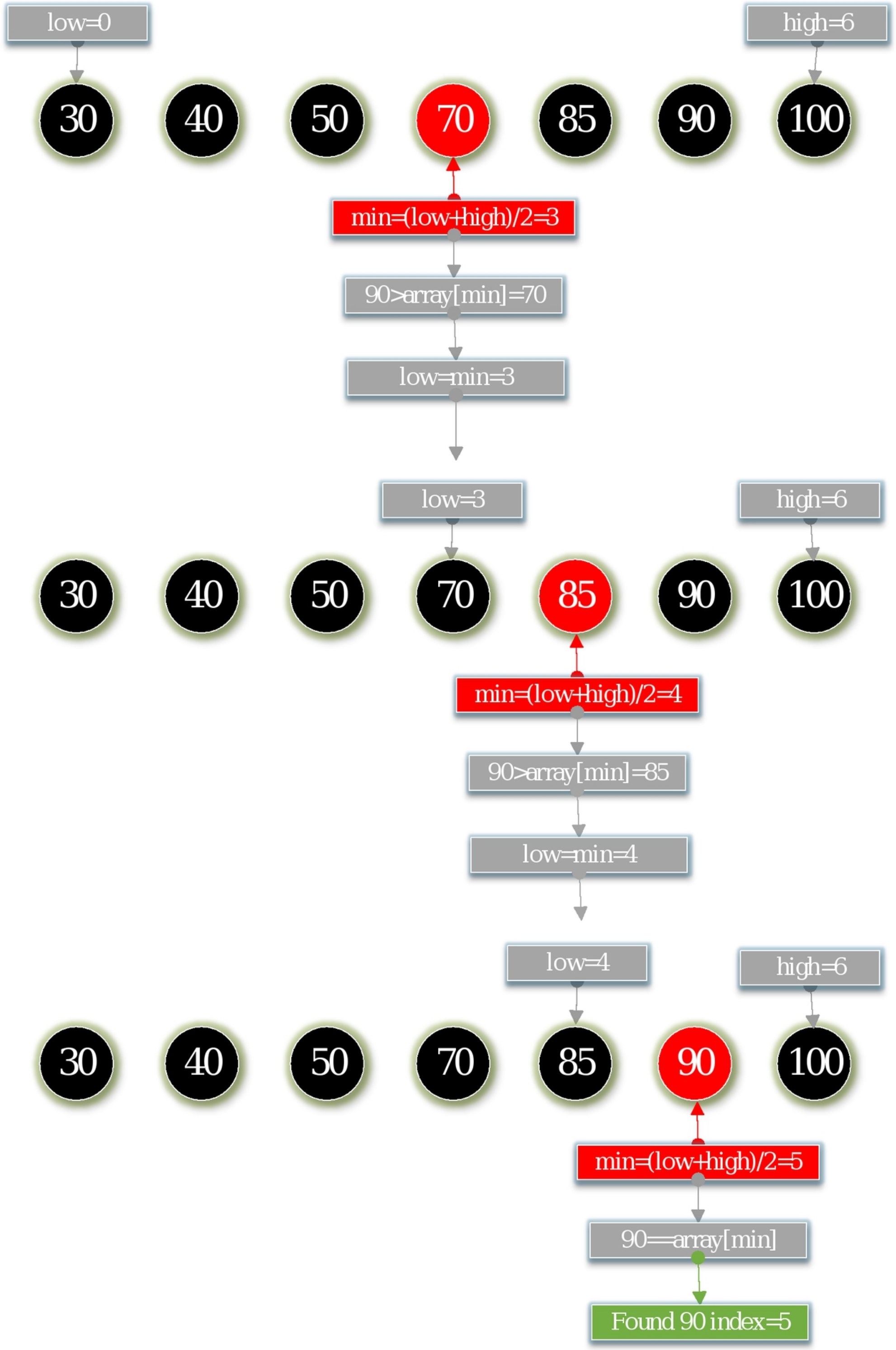
30
40
50
70
85
90
100

1. Initialize the lowest index low=0 , the highest index high=scores.length-1
2. Find the searchValue of the middle index mid=(low+high)/2 scores[mid]
3. Compare the scores[mid] with searchValue
   If the scores[mid]==searchValue print current mid index,
   If scores[mid]>searchValue that the searchValue will be found between low and mid-1
4. And so on. Repeat step 3 until you find searchValue or low>=high to terminate the loop.

**Example 1 : Find the index of searchValue=40 in the array that has been sorted below.**


30
40
50
70
85
90
100
low=0
high=6
30
40
50
70
85
90
100
min=(low+high)/2=3
40<array[min]=70
high=min=3
low=0
high=3
30
40
50
70
85
90
100
min=(low+high)/2=1
40==array[min]
Found 40 index=1

**Example 2 : Find the index of searchValue=90 in the array that has been sorted below.**

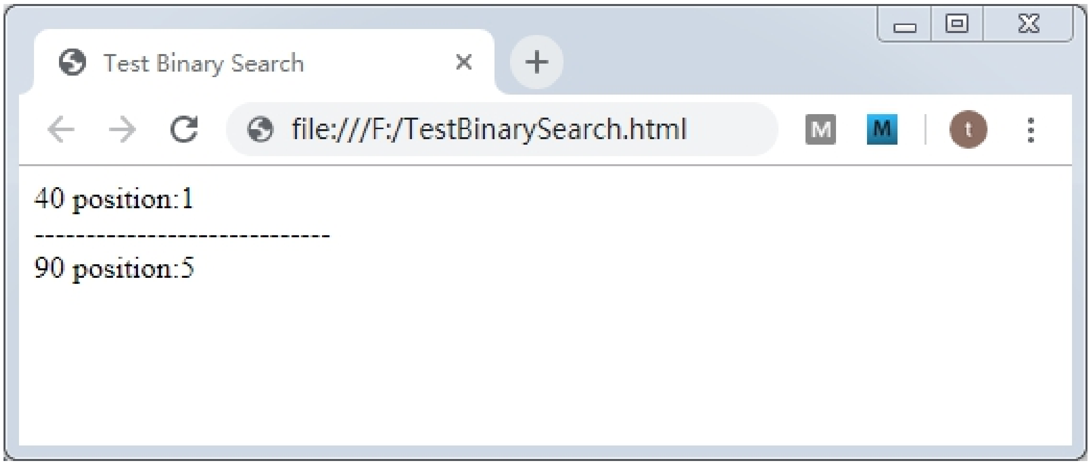

low=0
high=6
30
40
50
70
85
90
100
min=(low+high)/2=3
90>array[min]=70
low=min=3
low=3
high=6
30
40
50
70
85
90
100
min=(low+high)/2=4
90>array[min]=85
low=min=4
low=4
high=6
30
40
50
70
85
90
100
min=(low+high)/2=5
90==array[min]
Found 90 index=5

**1. Create a TestBinarySearch.html with Notepad and open it in your browser.**

```
<script type ="text/javascript" >
  class BinarySearch{
    static search( arrays, searchValue) {
      var low = 0 ;
      var high = arrays.length - 1 ;
      var mid = 0 ;
      while ( low <= high) {
        mid = ( low + high) / 2 ;
        if ( arrays[ mid] == searchValue) {
          return mid;
        } else if ( arrays[ mid] < searchValue) {
          low = mid + 1 ;
        } else if ( arrays[ mid] > searchValue) {
          high = mid - 1 ;
        }
      }
      return - 1 ;
    }
  }

  //////////////////////testing////////////////////

  var scores = [ 30 , 40 , 50 , 70 , 85 , 90 , 100 ];

  var searchValue = 40 ;
  var position = BinarySearch.search( scores, searchValue);
  document.write( searchValue + " position:" + position);

  document.write( "<br>---------------------------<br>" );

  searchValue = 90 ;
  position = BinarySearch.search( scores, searchValue);
  document.write( searchValue + " position:" + position);
```

```
</script>
```

## Result:

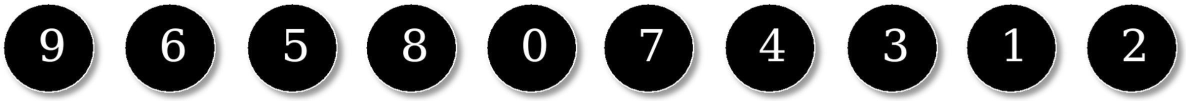

Test Binary Search
file:///F:/TestBinarySearch.html

```
40 position:1
------------------------------
90 position:5
```

# Shell Sorting

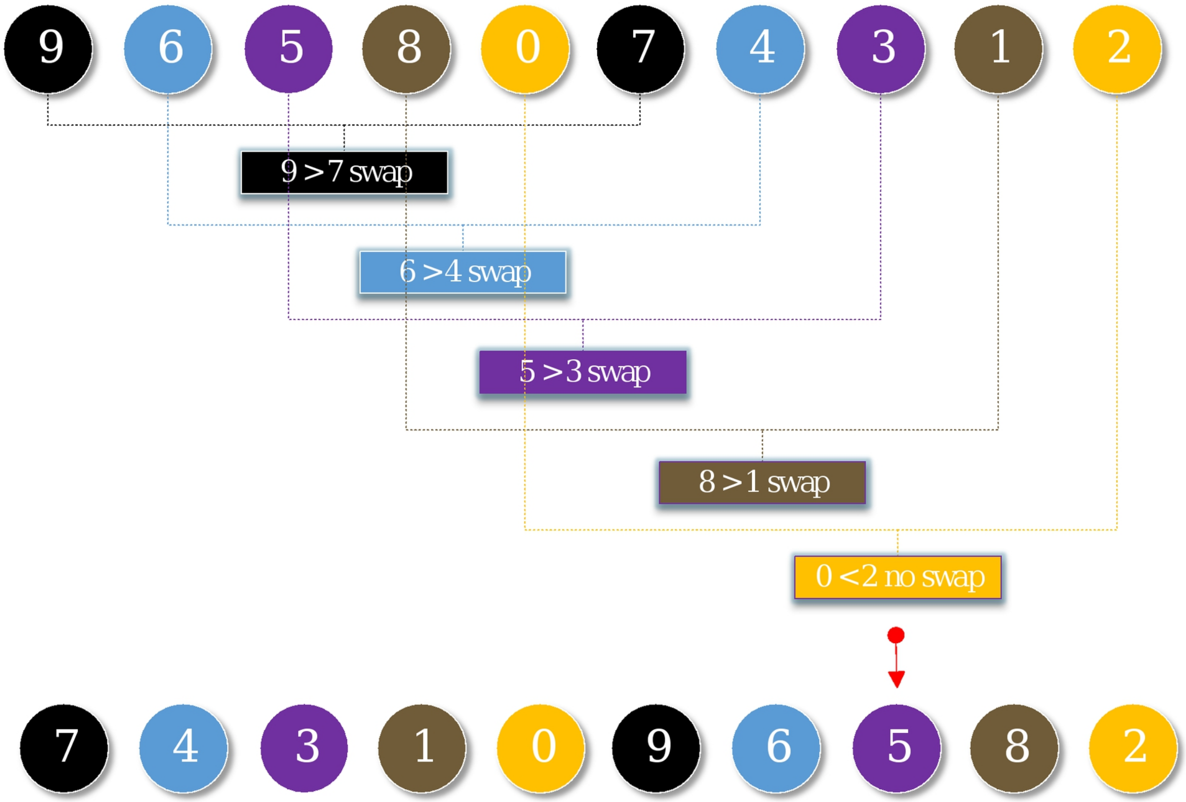
**Shell Sorting:**
An efficient insertion sort.

**Sort the following numbers from small to large by Shell Sorting**


9
6
5
8
0
7
4
3
1
2

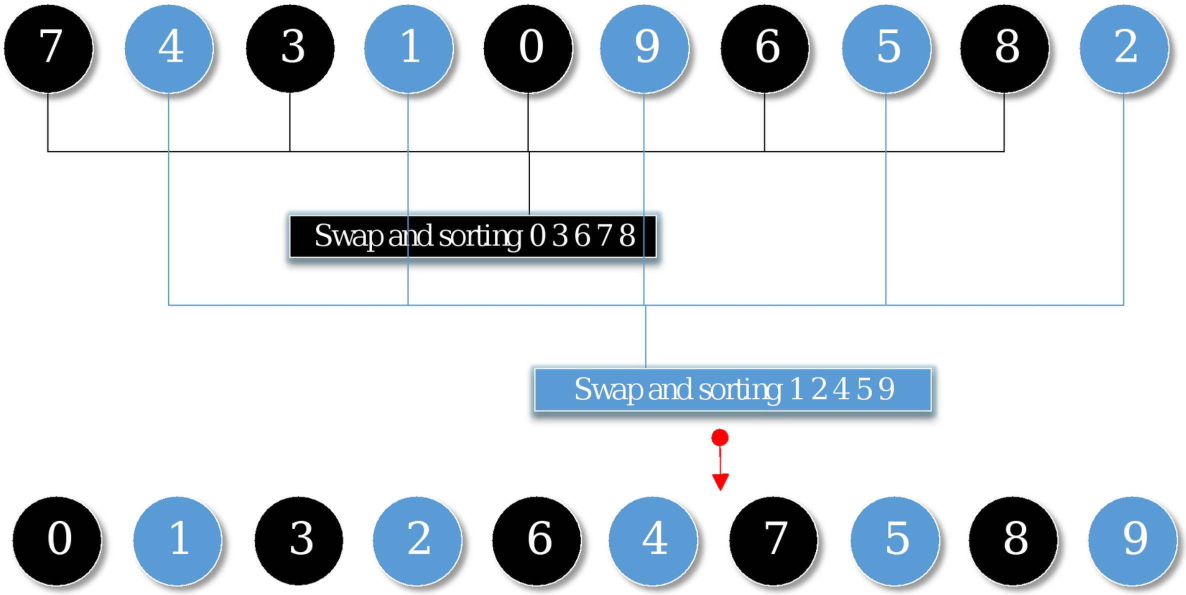
**Algorithmic result:**
The array is grouped according to a certain increment of subscripts, and the insertion of each group is sorted. As the increment decreases gradually until the increment is 1, the whole data is grouped and sorted.

**1． The first sorting：**
**gap = array.length / 2 = 5**


9
6
5
8
0
7
4
3
1
2
9 >7 swap
6 >4 swap
5 >3 swap
8 >1 swap
0 <2 no swap
7
4
3
1
0
9
6
5
8
2

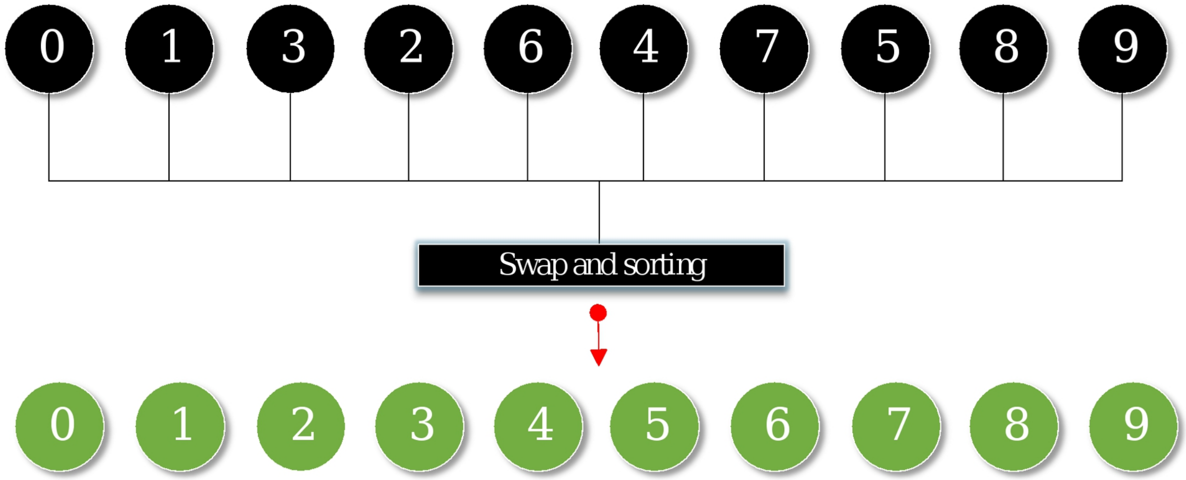
## 2 . The second sorting :

**gap = 5 / 2 = 2**


7 4 3 1 0 9 6 5 8 2
Swap and sorting 0 3 6 7 8
Swap and sorting 1 2 4 5 9
0 1 3 2 6 4 7 5 8 9

### 3 . The third sorting :

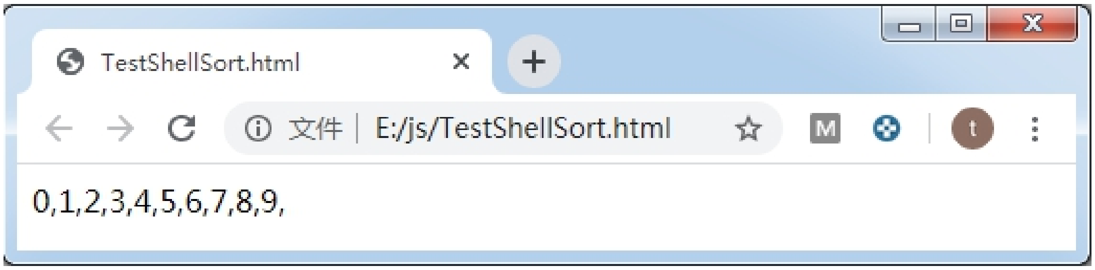
**gap = 2 / 2 = 1**


0
1
3
2
6
4
7
5
8
9
Swap and sorting
0
1
2
3
4
5
6
7
8
9

**1. Create a TestShellSort.html with Notepad and open it in your browser.**

```
<script type ="text/javascript" >
  function shellSort( array) {
      var middle = parseInt( array.length / 2 );
      for ( var gap = middle ; gap > 0 ; gap = parseInt( gap / 2 )) {
         for ( var i = gap; i < array.length; i++) {
            var j = i;
            while ( j - gap >= 0 && array[ j] < array[ j - gap]) {
               swap( array, j, j - gap);
               j = j - gap;
            }
         }
      }
  }

  function swap( array, a, b) {
      array[ a] = array[ a] + array[ b];
      array[ b] = array[ a] - array[ b];
      array[ a] = array[ a] - array[ b];
  }

  ////////////////////testing//////////////////

  var scores = [ 9 , 6 , 5 , 8 , 0 , 7 , 4 , 3 , 1 , 2 ];
  shellSort( scores);
  for ( var i = 0 ; i < scores.length; i++) {
      document.write( scores[ i] + "," );
  }

</script>
```

**Result:**

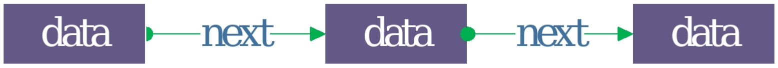


TestShellSort.html
文件
E:/js/TestShellSort.html

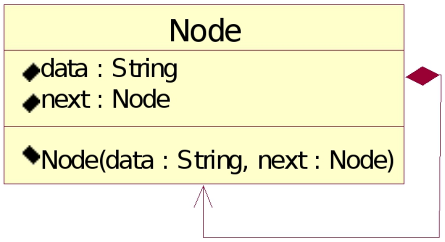
```
0,1,2,3,4,5,6,7,8,9,
```

# Unidirectional Linked List

**Unidirectional Linked List:**
Is a chained storage structure of a linear table, which is connected by a node. Each node consists of data and next pointer to the next node.


data
next
data
next
data

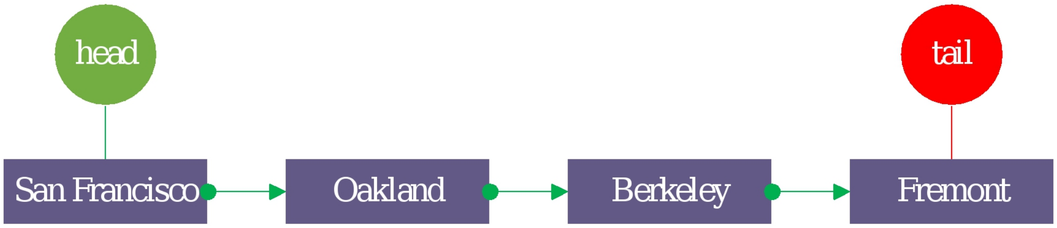
**UML Diagram**


Node
data : String
next : Node
Node(data : String, next : Node)

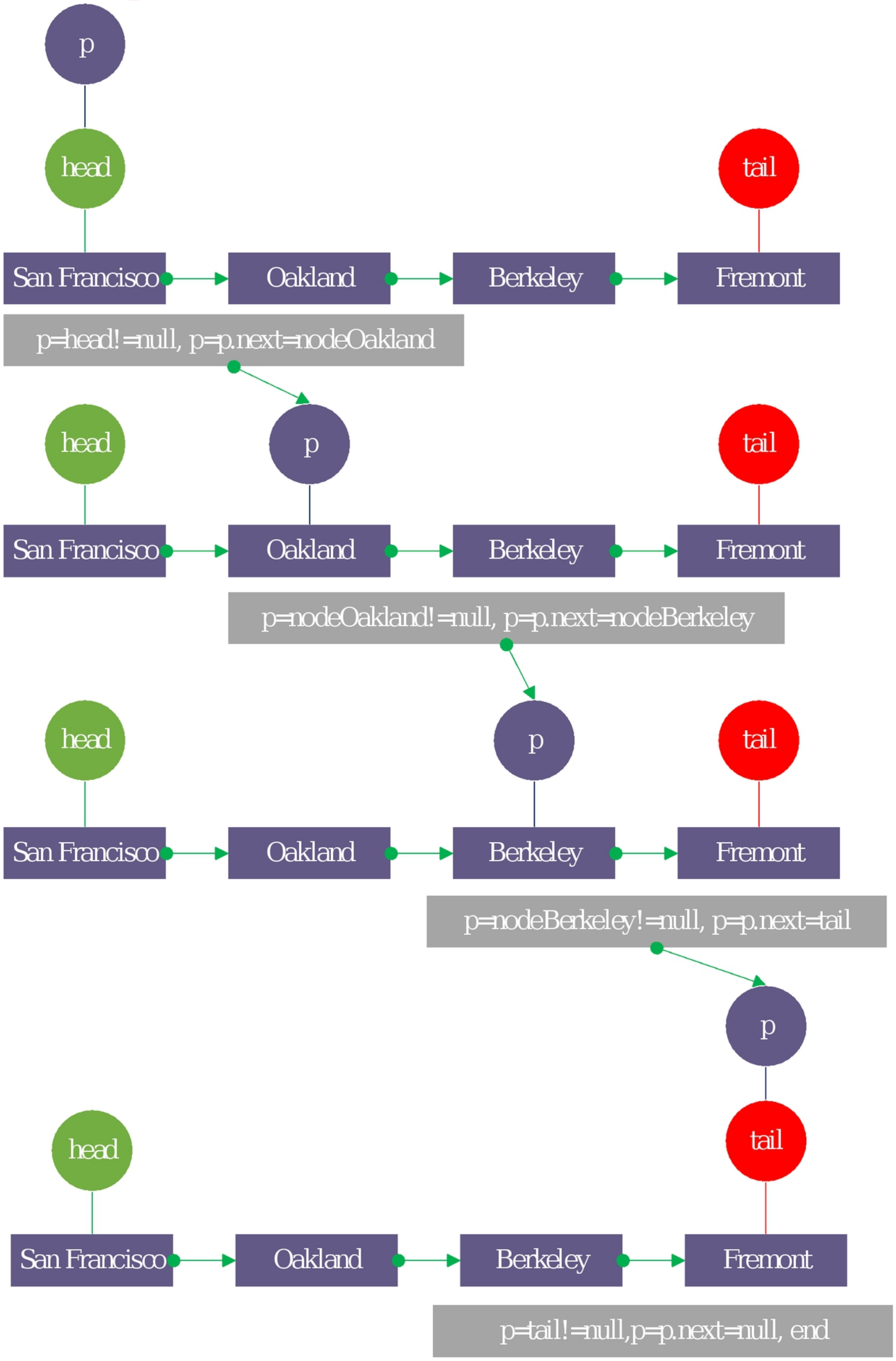
```
class Node{
    constructor( data, next){
        this.data = data;
        this.next = next;
    }

    getData(){
        return this.data;
    }
}
```

## 1. Unidirectional Linked List initialization .

**Example : Construct a San Francisco subway Unidirectional linked list**


head
tail
San Francisco
Oakland
Berkeley
Fremont

### 2. traversal output .

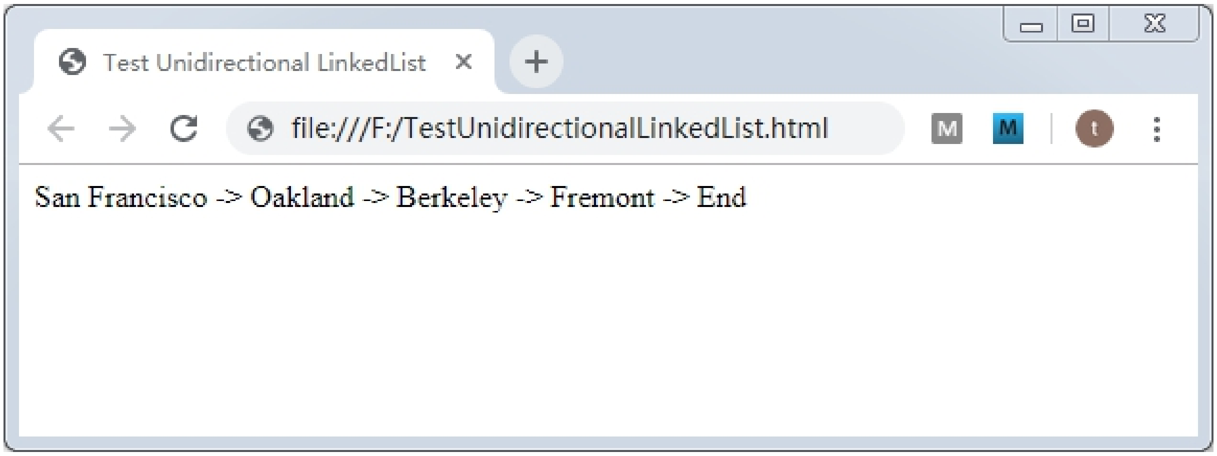

p
head
tail
San Francisco
Oakland
Berkeley
Fremont
p=head!=null, p=p.next=nodeOakland
head
p
tail
San Francisco
Oakland
Berkeley
Fremont
p=nodeOakland!=null, p=p.next=nodeBerkeley
head
p
tail
San Francisco
Oakland
Berkeley
Fremont
p=nodeBerkeley!=null, p=p.next=tail
p
head
tail
San Francisco
Oakland
Berkeley
Fremont
p=tail!=null,p=p.next=null, end

### 1. Create a TestUnidirectionalLinkedList.html with Notepad and open it in your browser.

```
<script type ="text/javascript" >
```

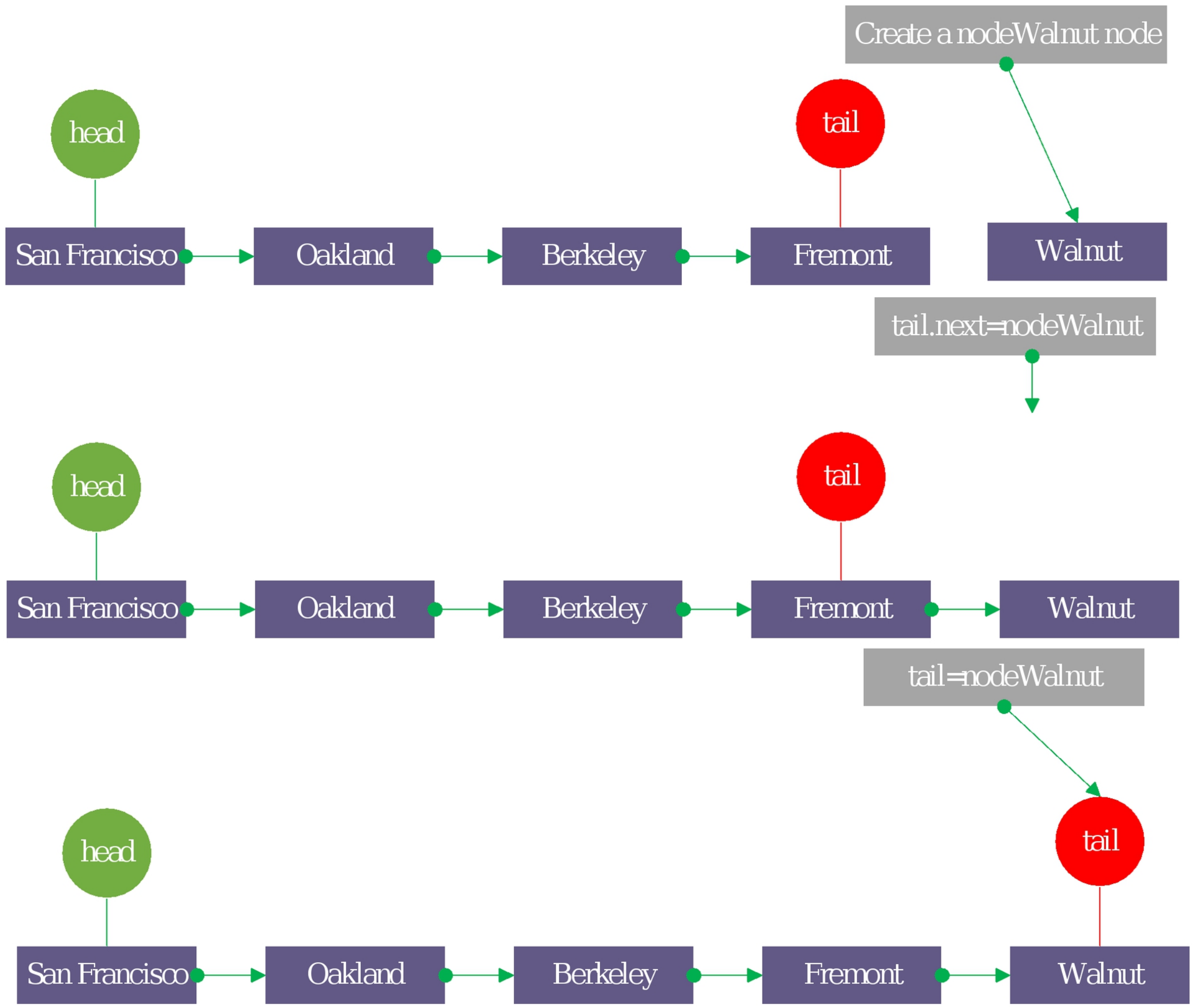
```
class Node{
    constructor( data, next){
        this.data = data;
        this.next = next;
    }

    getData(){
        return this.data;
    }
}

class LinkedList{
    init() {
        // the first node called head node
        this.head = new Node( "San Francisco" , null );

        var nodeOakland = new Node( "Oakland" , null );
        this.head.next = nodeOakland;

        var nodeBerkeley = new Node( "Berkeley" , null );
        nodeOakland.next = nodeBerkeley;

        // the last node called tail node
        this.tail = new Node( "Fremont" , null );
        nodeBerkeley.next = this.tail;

        return this.head;
    }

    print( node) {
        var p = node;
        while ( p != null ) // From the beginning to the end
        {
            var data = p.getData();
            document.write( data + " -> " );
            p = p.next;
        }
        document.write( "End<br><br>" );
    }
  }

  ////////////////////testing/////////////////

  var linkedList = new LinkedList();
  var head = linkedList.init();
  linkedList.print( head);
</script>
```

**Result:**


Test Unidirectional LinkedList
file:///F:/TestUnidirectionalLinkedList.html

```
San Francisco -> Oakland -> Berkeley -> Fremont -> End
```

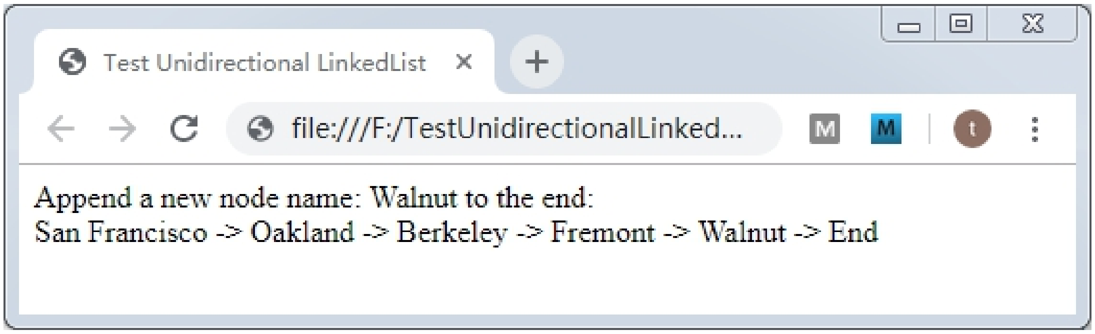
## 3. Append a new node name: Walnut  to the end.


Create a nodeWalnut node
head
tail
San Francisco
Oakland
Berkeley
Fremont
Walnut
tail.next=nodeWalnut
head
tail
San Francisco
Oakland
Berkeley
Fremont
Walnut
tail=nodeWalnut
head
tail
San Francisco
Oakland
Berkeley
Fremont
Walnut

### 1. Create a TestUnidirectionalLinkedListAppend.html with Notepad

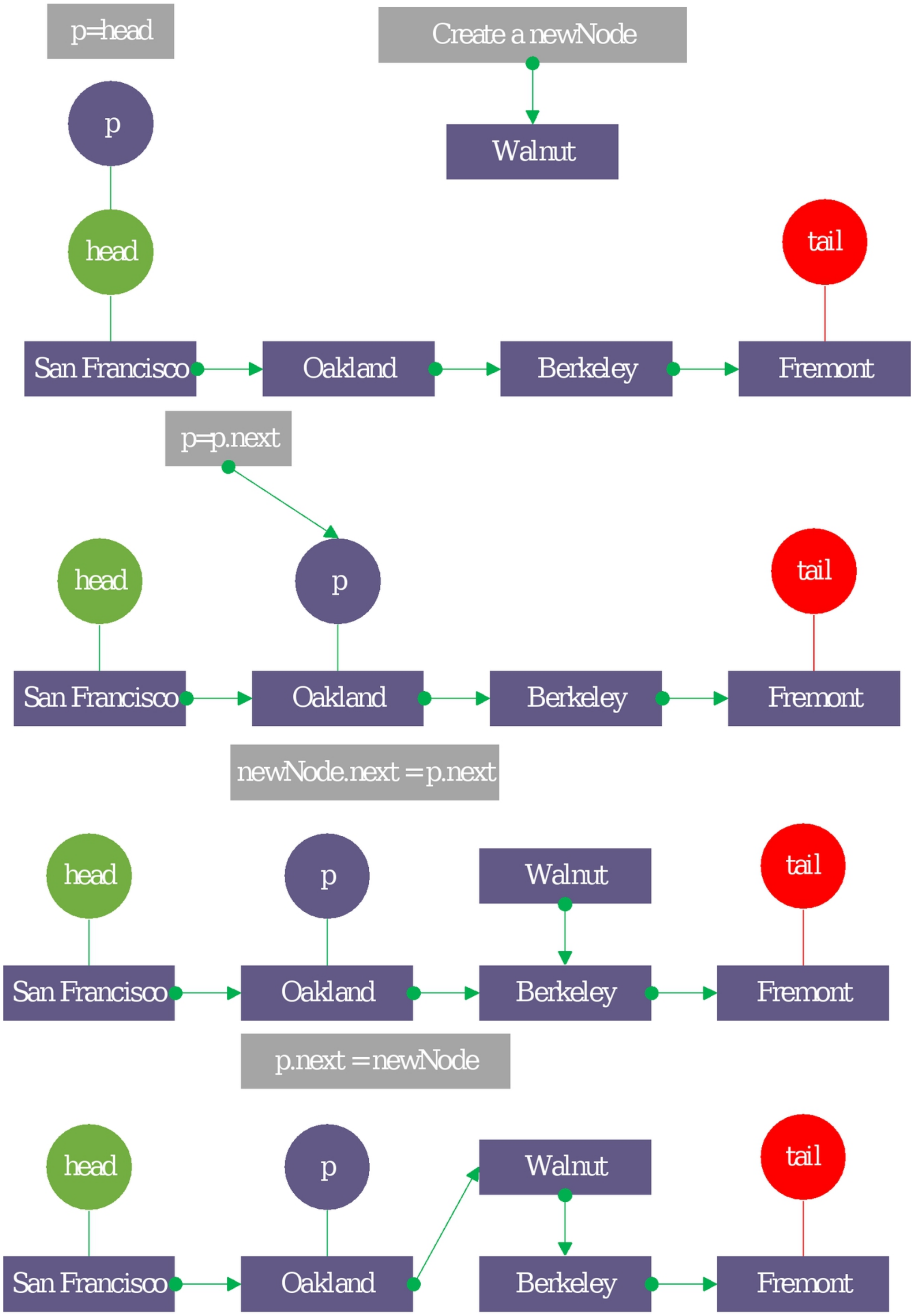
```
<script type ="text/javascript" >
    class Node{
        constructor( data, next){
            this.data = data;
            this.next = next;
        }

        getData(){
            return this.data;
        }
    }

    class LinkedList{

        init() {
            // the first node called head node
            this.head = new Node( "San Francisco" , null );

            var nodeOakland = new Node( "Oakland" , null );
            this.head.next = nodeOakland;

            var nodeBerkeley = new Node( "Berkeley" , null );
            nodeOakland.next = nodeBerkeley;

            // the last node called tail node
            this.tail = new Node( "Fremont" , null );
            nodeBerkeley.next = this.tail;

            return this.head;
        }

        add( newNode) {
            this.tail.next = newNode;
            this.tail = newNode;
        }








        print( node) {
            var p = node;
            while ( p != null ) // From the beginning to the end
            {
                var data = p.getData();
                document.write( data + " -> " );
                p = p.next;
            }
            document.write( "End<br><br>" );
        }
    }

    //////////////////////testing////////////////////

    var linkedList = new LinkedList();
    var head = linkedList.init();

    document.write( "Append a new node name: Walnut  to the end: <br>" );
    linkedList.add( new Node( "Walnut" , null ));

    linkedList.print( head);
</script>
```

**Result:**


Test Unidirectional LinkedList
file:///F:/TestUnidirectionalLinked...

```
Append a new node name: Walnut to the end:
San Francisco -> Oakland -> Berkeley -> Fremont -> Walnut -> End
```

### 3. Insert a node Walnut in position 2.

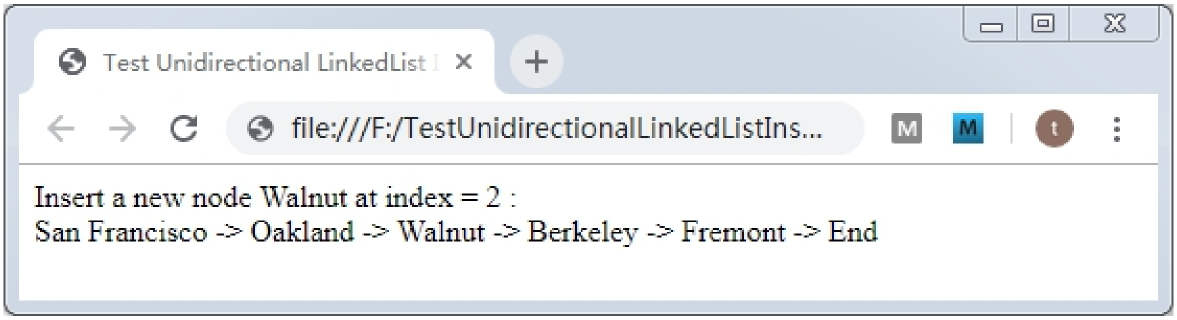

p=head
Create a newNode
p
Walnut
head
tail
San Francisco
Oakland
Berkeley
Fremont
p=p.next
head
p
tail
San Francisco
Oakland
Berkeley
Fremont
newNode.next = p.next
head
p
Walnut
tail
San Francisco
Oakland
Berkeley
Fremont
p.next = newNode
head
p
Walnut
tail
San Francisco
Oakland
Berkeley
Fremont

### 1. Create a TestUnidirectionalLinkedListInsert.html with Notepad

```
<script type ="text/javascript" >
  class Node{
    constructor( data, next){
      this.data = data;
      this.next = next;
    }

    getData(){
      return this.data;
    }
  }

  class LinkedList{
    init() {
      // the first node called head node
      this.head = new Node( "San Francisco" , null );

      var nodeOakland = new Node( "Oakland" , null );
      this.head.next = nodeOakland;

      var nodeBerkeley = new Node( "Berkeley" , null );
      nodeOakland.next = nodeBerkeley;

      // the last node called tail node
      this.tail = new Node( "Fremont" , null );
      nodeBerkeley.next = this.tail;

      return this.head;
    }

    insert( insertPosition, newNode) {
      var p = this.head;
      var i = 0 ;
      // Move the node to the insertion position
      while ( p.next != null && i < insertPosition - 1 ) {
```

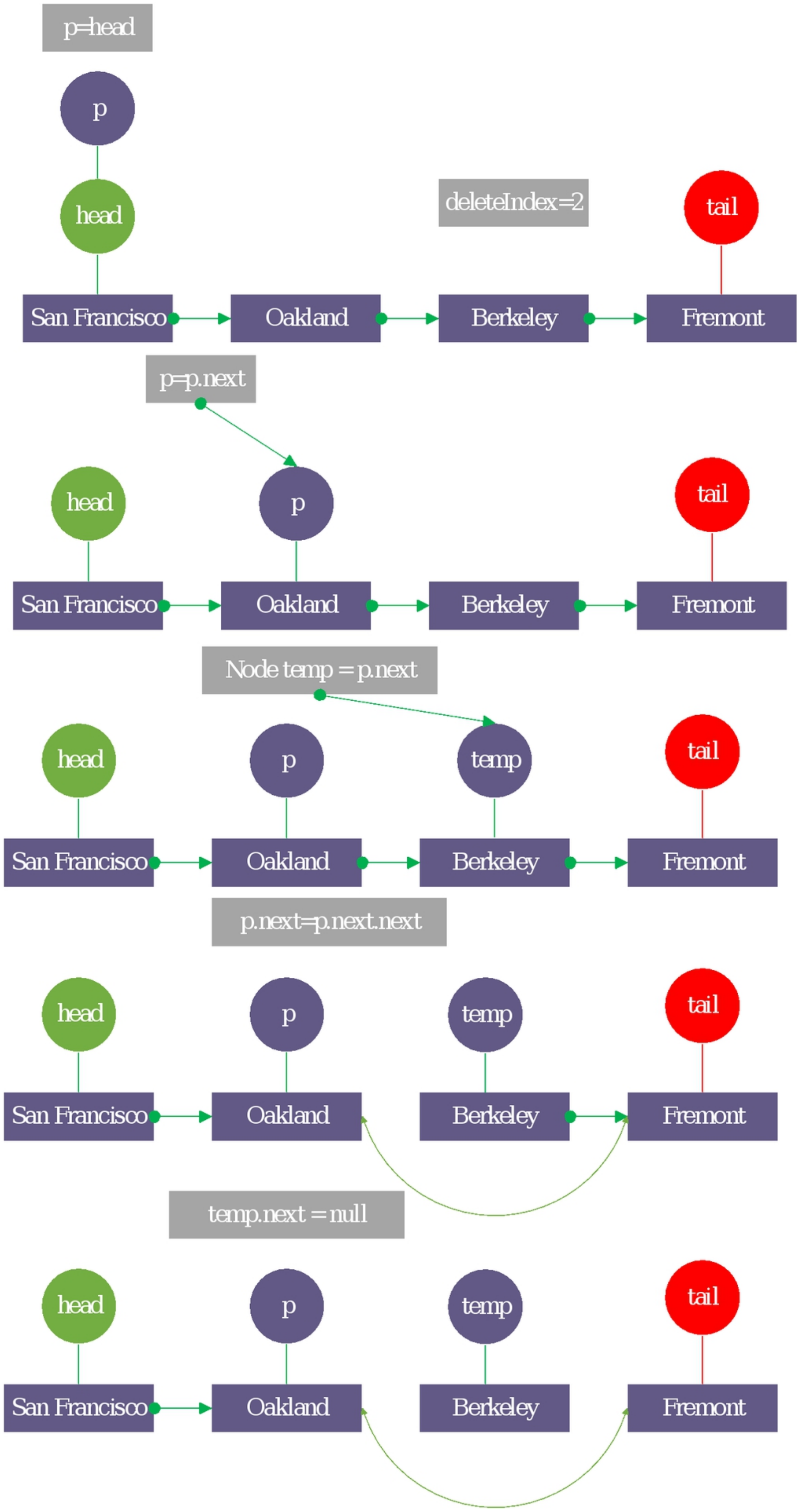
```
                p = p.next;
                i++;
            }
            newNode.next = p.next; // newNode next point to next node
            p.next = newNode; // current next point to newNode
        }

        print( node) {
            var p = node;
            while ( p != null ) // From the beginning to the end
            {
                var data = p.getData();
                document.write( data + " -> " );
                p = p.next;
            }
            document.write( "End<br><br>" );
        }
    }

    //////////////////////testing////////////////////

    var linkedList = new LinkedList();
    var head = linkedList.init();

    document.write( "Insert a new node Walnut at index = 2 : <br>" );
    linkedList.insert( 2 , new Node( "Walnut" , null ));

    linkedList.print( head);
</script>
```

**Result:**


Test Unidirectional LinkedList I
file:///F:/TestUnidirectionalLinkedListIns...

```
Insert a new node Walnut at index = 2 :
San Francisco -> Oakland -> Walnut -> Berkeley -> Fremont -> End
```

## 4. Delete the index=2 node.

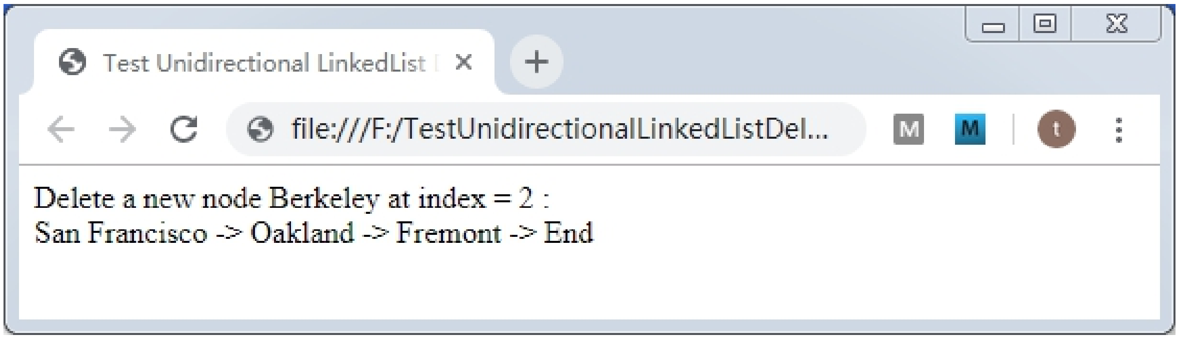

p=head
p
head
deleteIndex=2
tail
San Francisco
Oakland
Berkeley
Fremont
p=p.next
head
p
tail
San Francisco
Oakland
Berkeley
Fremont
Node temp = p.next
head
p
temp
tail
San Francisco
Oakland
Berkeley
Fremont
p.next=p.next.next
head
p
temp
tail
San Francisco
Oakland
Berkeley
Fremont
temp.next = null
head
p
temp
tail
San Francisco
Oakland
Berkeley
Fremont

## 1. Create a TestUnidirectionalLinkedListDelete.html with Notepad

```
<script type ="text/javascript" >
  class Node{
    constructor( data, next){
        this.data = data;
        this.next = next;
    }

    getData(){
        return this.data;
    }
}

class LinkedList{
    init() {
        // the first node called head node
        this.head = new Node( "San Francisco" , null );

        var nodeOakland = new Node( "Oakland" , null );
        this.head.next = nodeOakland;

        var nodeBerkeley = new Node( "Berkeley" , null );
        nodeOakland.next = nodeBerkeley;

        // the last node called tail node
        this.tail = new Node( "Fremont" , null );
        nodeBerkeley.next = this.tail;

        return this.head;
    }

    remove( removePosition) {
        var p = this.head;
        var i = 0 ;
        // Move the node to the previous node position that was deleted
        while ( p.next != null && i < removePosition - 1 ) {
            p = p.next;
            i++;
        }
        var temp = p.next; // Save the node you want to delete
```

```
            p.next = p.next.next; // Previous node next points to next of delete the node
            temp.next = null ;
        }

        print( node) {
            var p = node;
            while ( p != null ) // From the beginning to the end
            {
                var data = p.getData();
                document.write( data + " -> " );
                p = p.next;
            }
            document.write( "End<br><br>" );
        }
    }

    ////////////////////testing//////////////////

    var linkedList = new LinkedList();
    var head = linkedList.init();

    document.write( "Delete a new node Berkeley at index = 2 : <br>" );
    linkedList.remove( 2 );

    linkedList.print( head);
</script>
```

**Result:**

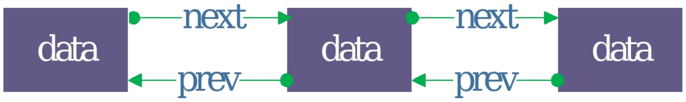

Test Unidirectional LinkedList
file:///F:/TestUnidirectionalLinkedListDel...

```
Delete a new node Berkeley at index = 2 :
San Francisco -> Oakland -> Fremont -> End
```

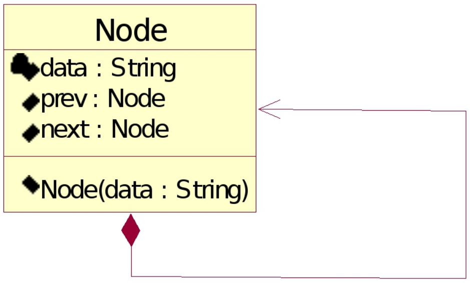
# Doubly Linked List

**Doubly Linked List:**

It is a chained storage structure of a linear table. It is connected by nodes in two directions. Each node consists of data, pointing to the previous node and pointing to the next node.


data
next
prev
data
next
prev
data

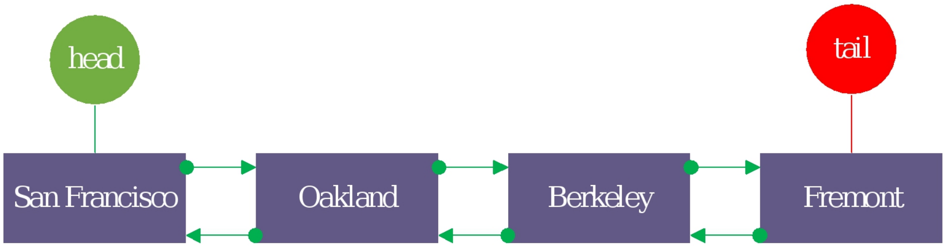
**UML Diagram**


Node
data : String
prev : Node
next : Node
Node(data : String)

```
class Node{
    constructor( data, prev, next){
        this.data = data;
        this.prev = prev;
        this.next = next;
    }

    getData(){
        return this.data;
    }
}
```

## 1. Doubly Linked List initialization .

**Example : Construct a San Francisco subway Doubly linked list**

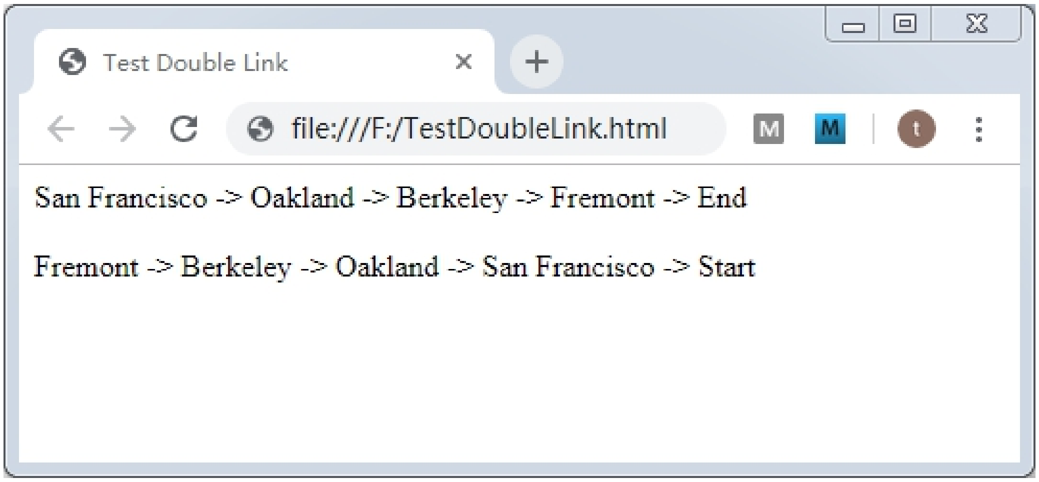

head
tail
San Francisco
Oakland
Berkeley
Fremont

## 2. Create a TestDoubleLink.html with Notepad

```
<script type ="text/javascript" >
   class Node{
      constructor( data, prev, next){
         this.data = data;
         this.prev = prev;
         this.next = next;
      }

      getData(){
         return this.data;
      }
   }
```

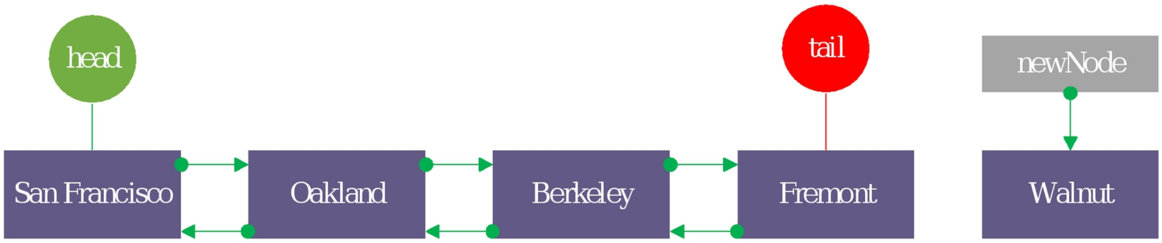
```
class DoubleLinkList{
    init() {
        this.head = new Node( "San Francisco" );
        this.head.prev = null ;
        this.head.next = null ;

        var nodeOakland = new Node( "Oakland" );
        nodeOakland.prev = this.head;
        nodeOakland.next = null ;
        this.head.next = nodeOakland;

        var nodeBerkeley = new Node( "Berkeley" );
        nodeBerkeley.prev = nodeOakland;
        nodeBerkeley.next = null ;
        nodeOakland.next = nodeBerkeley;

        this.tail = new Node( "Fremont" );
        this.tail.prev = nodeBerkeley;
        this.tail.next = null ;
        nodeBerkeley.next = this.tail;
        return this.head;
    }

    print( node) {
        var p = node;
        var end = null ;
        while ( p != null )
        {
            var data = p.getData();
            document.write( data + " -> " );
            end = p;
            p = p.next;
            }
            document.write( "End <br><br>" );

            p = end;
            while ( p != null )
            {
                var data = p.getData();
                document.write( data + " -> " );
                p = p.prev;
            }
            document.write( "Start<br><br>" );
        }
    }

    ////////////////////testing/////////////////
    var doubleLinkList = new DoubleLinkList();
    var head = doubleLinkList.init();
    doubleLinkList.print( head);
</script>
```

**Result:**

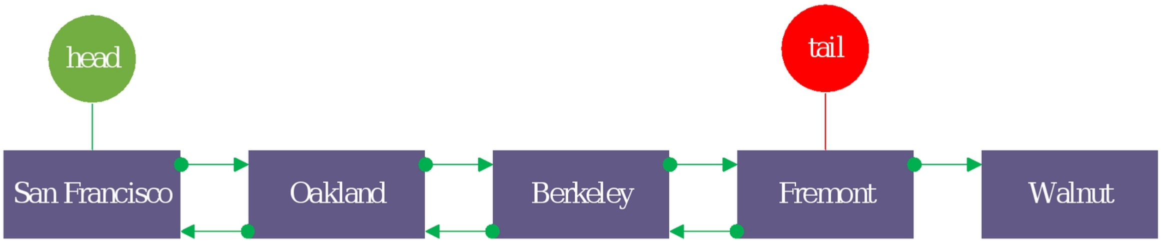

Test Double Link
file:///F:/TestDoubleLink.html

```
San Francisco -> Oakland -> Berkeley -> Fremont -> End

Fremont -> Berkeley -> Oakland -> San Francisco -> Start
```

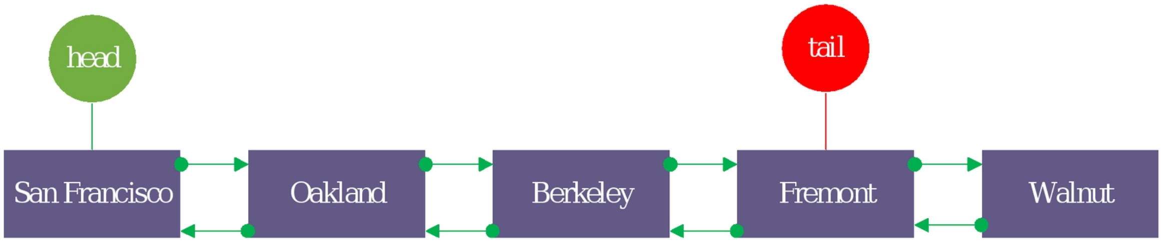
## 3. add a node Walnut at the end of Fremont.


head
tail
newNode
San Francisco
Oakland
Berkeley
Fremont
Walnut

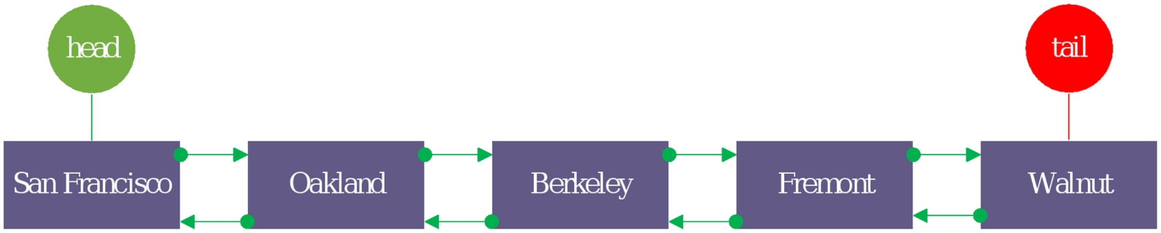
tail.next = newNode


head
tail
San Francisco
Oakland
Berkeley
Fremont
Walnut

newNode.prev = tail


head
tail
San Francisco
Oakland
Berkeley
Fremont
Walnut

tail = newNode

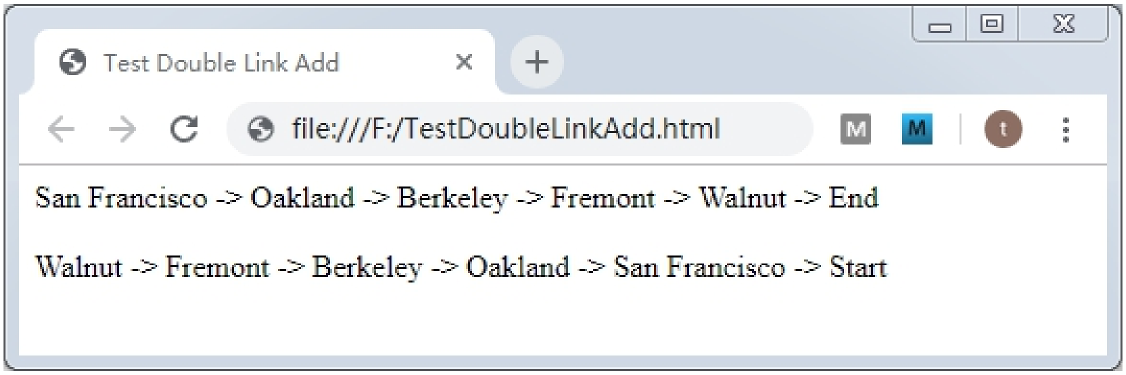

head
tail
San Francisco
Oakland
Berkeley
Fremont
Walnut

**Create a TestDoubleLinkAdd.html with Notepad**

```
<script type ="text/javascript" >
   class Node{
      constructor( data, prev, next){
         this.data = data;
         this.prev = prev;
         this.next = next;
      }

      getData(){
         return this.data;
      }
   }

   class DoubleLinkList{
      init() {
         this.head = new Node( "San Francisco" );
         this.head.prev = null ;
         this.head.next = null ;

         var nodeOakland = new Node( "Oakland" );
         nodeOakland.prev = this.head;
         nodeOakland.next = null ;
         this.head.next = nodeOakland;

         var nodeBerkeley = new Node( "Berkeley" );
         nodeBerkeley.prev = nodeOakland;
         nodeBerkeley.next = null ;
         nodeOakland.next = nodeBerkeley;

         this.tail = new Node( "Fremont" );
         this.tail.prev = nodeBerkeley;
         this.tail.next = null ;
         nodeBerkeley.next = this.tail;
         return this.head;
      }
```

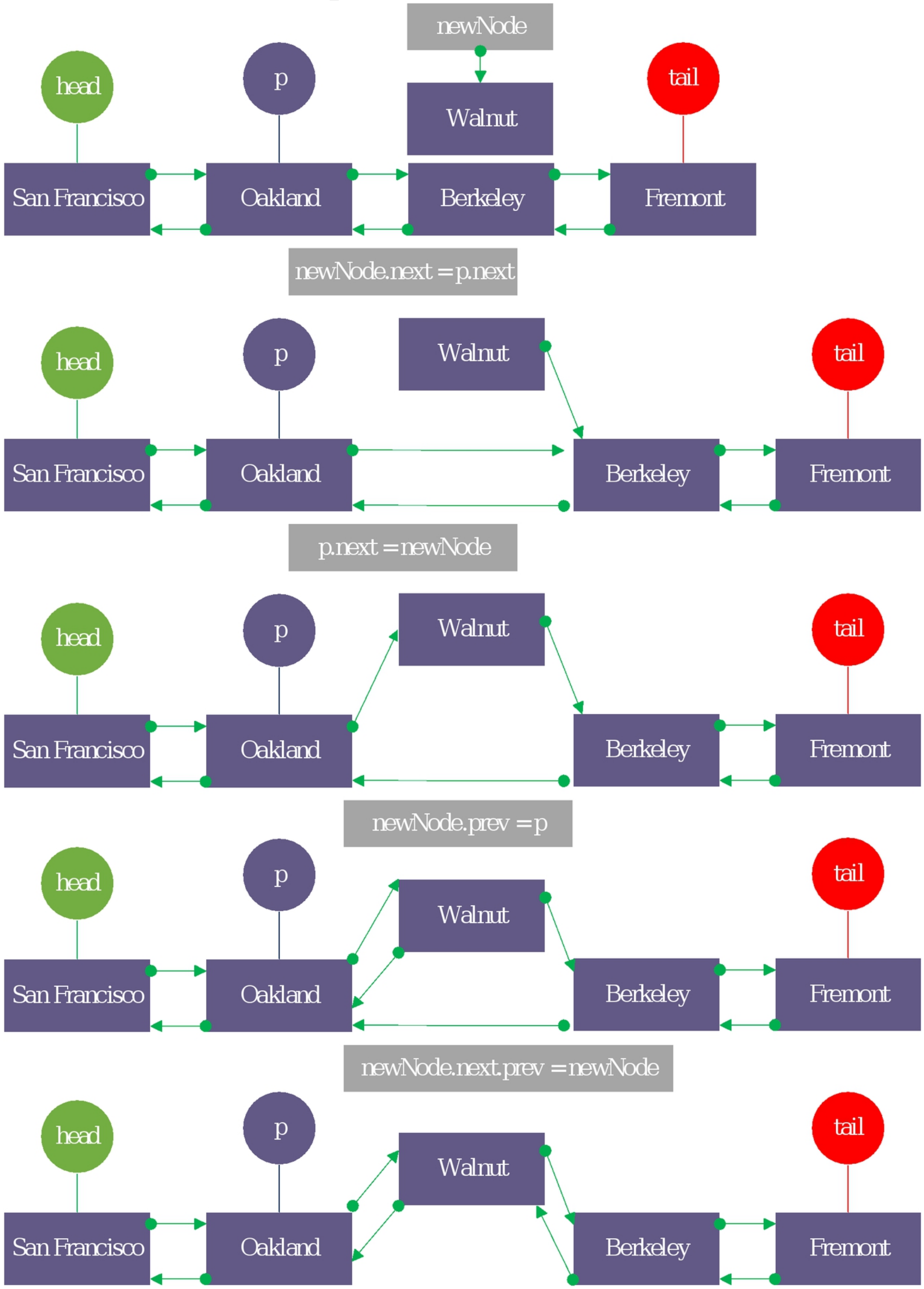
```
        add( newNode) {
            this.tail.next = newNode;
            newNode.prev = this.tail;
            this.tail = newNode;
        }

        print( node) {
            var p = node;
            var end = null ;
            while ( p != null )
            {
                var data = p.getData();
                document.write( data + " -> " );
                end = p;
                p = p.next;
            }
            document.write( "End <br><br>" );

            p = end;
            while ( p != null )
            {
                var data = p.getData();
                document.write( data + " -> " );
                p = p.prev;
            }
            document.write( "Start<br><br>" );
        }
    }

    //////////////////////testing////////////////////
    var doubleLinkList = new DoubleLinkList();
    var head = doubleLinkList.init();
    doubleLinkList.add( new Node( "Walnut" ));
    doubleLinkList.print( head);
</script>
```

**Result:**


Test Double Link Add
file:///F:/TestDoubleLinkAdd.html

```
San Francisco -> Oakland -> Berkeley -> Fremont -> Walnut -> End

Walnut -> Fremont -> Berkeley -> Oakland -> San Francisco -> Start
```

### 3. Insert a node Walnut in position 2.


newNode
head
p
Walnut
tail
San Francisco
Oakland
Berkeley
Fremont
newNode.next = p.next
head
p
Walnut
tail
San Francisco
Oakland
Berkeley
Fremont
p.next = newNode
head
p
Walnut
tail
San Francisco
Oakland
Berkeley
Fremont
newNode.prev = p
head
p
Walnut
tail
San Francisco
Oakland
Berkeley
Fremont
newNode.next.prev = newNode
head
p
Walnut
tail
San Francisco
Oakland
Berkeley
Fremont

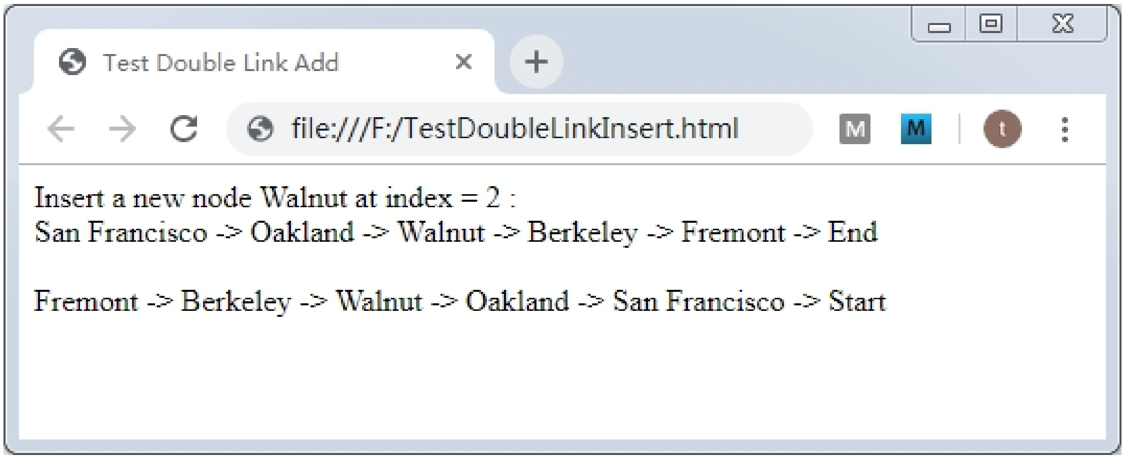
**Create a TestDoubleLinkInsert.html with Notepad**

```
<script type ="text/javascript" >
```

```
class Node{
    constructor( data, prev, next){
        this.data = data;
        this.prev = prev;
        this.next = next;
    }

    getData(){
        return this.data;
    }
}

class DoubleLinkList{

    init() {
        this.head = new Node( "San Francisco" );
        this.head.prev = null ;
        this.head.next = null ;

        var nodeOakland = new Node( "Oakland" );
        nodeOakland.prev = this.head;
        nodeOakland.next = null ;
        this.head.next = nodeOakland;

        var nodeBerkeley = new Node( "Berkeley" );
        nodeBerkeley.prev = nodeOakland;
        nodeBerkeley.next = null ;
        nodeOakland.next = nodeBerkeley;

        this.tail = new Node( "Fremont" );
        this.tail.prev = nodeBerkeley;
        this.tail.next = null ;
        nodeBerkeley.next = this.tail;
        return this.head;
    }
```

```
insert( insertPosition, newNode) {
    var p = this.head;
    var i = 0 ;
    // Move the node to the insertion position
    while ( p.next != null && i < insertPosition- 1 ) {
        p = p.next;
        i++;
    }

    newNode.next = p.next;
    p.next = newNode;
    newNode.prev = p;
    newNode.next.prev = newNode;
}

print( node) {
    var p = node;
    var end = null ;
    while ( p != null )
    {
        var data = p.getData();
        document.write( data + " -> " );
        end = p;
        p = p.next;
    }
    document.write( "End <br><br>" );

    p = end;
    while ( p != null )
    {
        var data = p.getData();
        document.write( data + " -> " );
        p = p.prev;
        }
        document.write( "Start<br><br>" );
    }
  }









  /////////////////////testing//////////////////
  var doubleLinkList = new DoubleLinkList();
  var head = doubleLinkList.init();

  document.write( "Insert a new node Walnut at index = 2 : <br>" );
  doubleLinkList.insert( 2 , new Node( "Walnut" ));

  doubleLinkList.print( head);
</script>
```

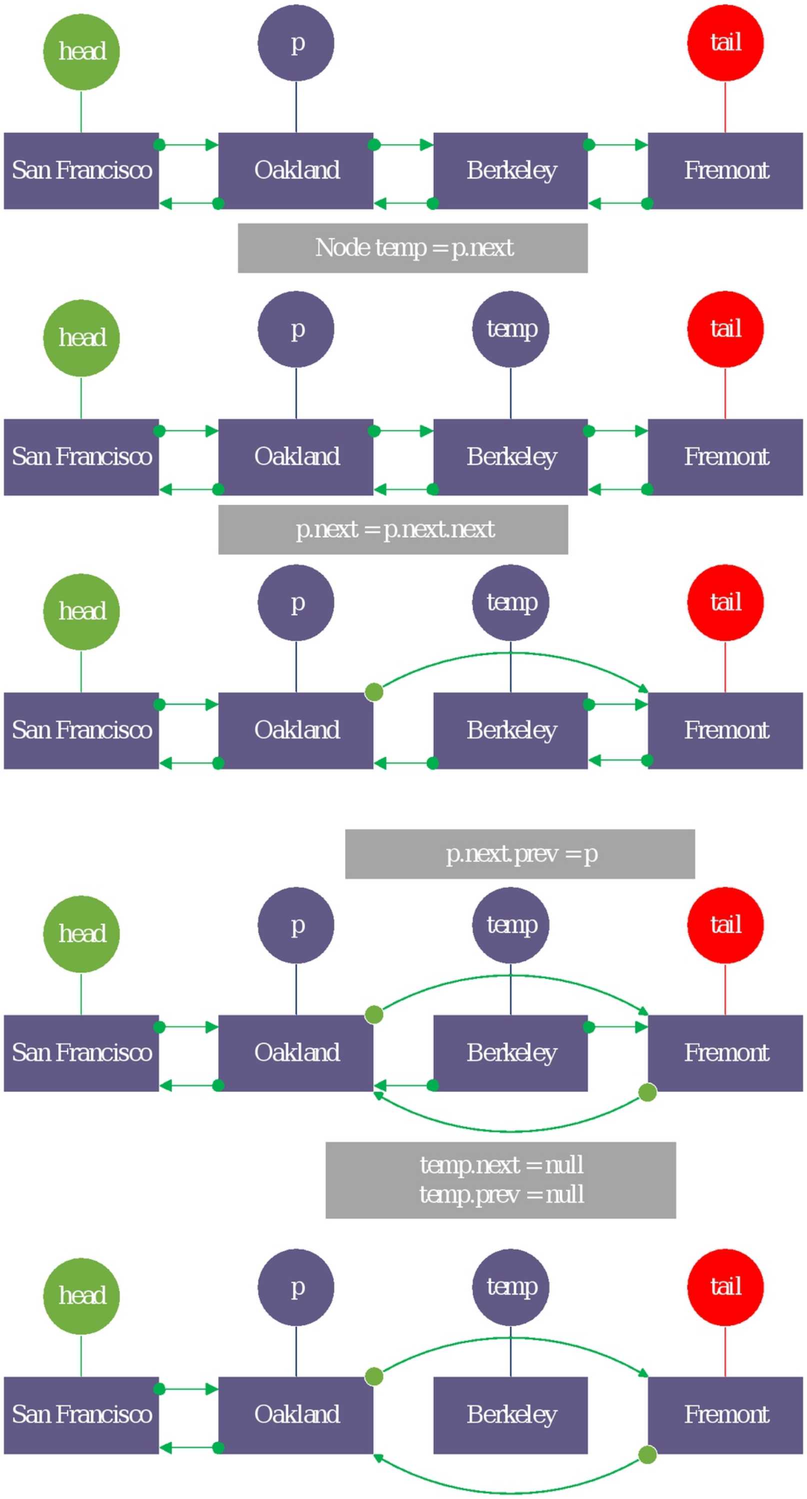
**Result:**



Test Double Link Add
file:///F:/TestDoubleLinkInsert.html

```
Insert a new node Walnut at index = 2 :
San Francisco -> Oakland -> Walnut -> Berkeley -> Fremont -> End

Fremont -> Berkeley -> Walnut -> Oakland -> San Francisco -> Start
```

### 4. Delete the index=2 node.


head
p
tail
San Francisco
Oakland
Berkeley
Fremont
temp

```
Node temp = p.next
```

```
p.next = p.next.next
```

```
p.next.prev = p
```

```
temp.next = null
temp.prev = null
```

### Create a TestDoubleLinkDelete.html with Notepad

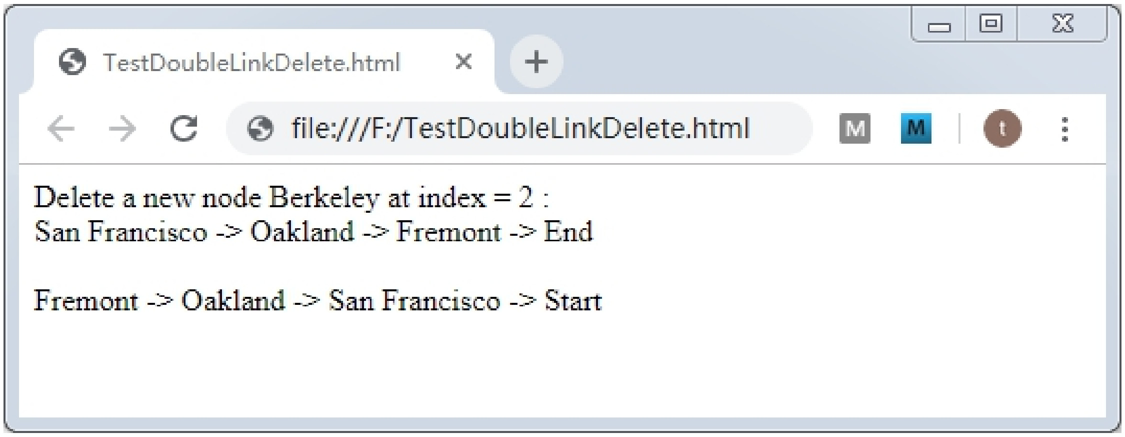
```
<script type ="text/javascript" >
```

```
class Node{
    constructor( data, prev, next){
        this.data = data;
        this.prev = prev;
        this.next = next;
    }

    getData(){
        return this.data;
    }
}

class DoubleLinkList{

    init() {
        this.head = new Node( "San Francisco" );
        this.head.prev = null ;
        this.head.next = null ;

        var nodeOakland = new Node( "Oakland" );
        nodeOakland.prev = this.head;
        nodeOakland.next = null ;
        this.head.next = nodeOakland;

        var nodeBerkeley = new Node( "Berkeley" );
        nodeBerkeley.prev = nodeOakland;
        nodeBerkeley.next = null ;
        nodeOakland.next = nodeBerkeley;

        this.tail = new Node( "Fremont" );
        this.tail.prev = nodeBerkeley;
        this.tail.next = null ;
        nodeBerkeley.next = this.tail;
        return this.head;
    }
```

```
remove( removePosition) {
    var p = this.head;
    var i = 0 ;
    // Move the node to the previous node  that was deleted
    while ( p.next != null && i < removePosition - 1 ) {
        p = p.next;
        i++;
    }

    var temp = p.next; // Save the node you want to delete
    p.next = p.next.next;
    p.next.prev = p;
    temp.next = null ; // Set the delete node next to null
    temp.prev = null ; // Set the delete node prev to null
}

print( node) {
    var p = node;
    var end = null ;
    while ( p != null )
    {
        var data = p.getData();
        document.write( data + " -> " );
        end = p;
        p = p.next;
    }
    document.write( "End <br><br>" );

    p = end;
    while ( p != null )
    {
        var data = p.getData();
        document.write( data + " -> " );
```

```
            p = p.prev;
        }
        document.write( "Start<br><br>" );
    }
}







//////////////////////testing//////////////////
var doubleLinkList = new DoubleLinkList();
var head = doubleLinkList.init();

document.write( "Delete a new node Berkeley at index = 2 : <br>" );
doubleLinkList.remove( 2 );

doubleLinkList.print( head);
</script>
```

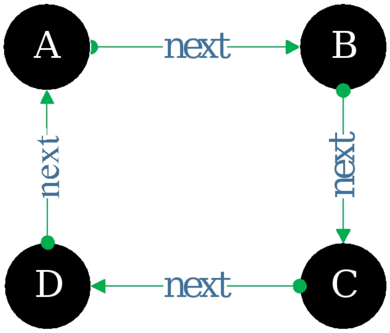
**Result:**


TestDoubleLinkDelete.html
file:///F:/TestDoubleLinkDelete.html

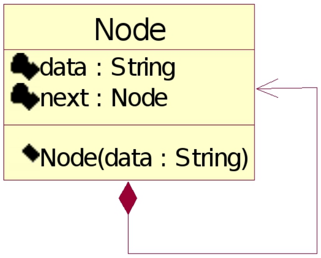
```
Delete a new node Berkeley at index = 2 :
San Francisco -> Oakland -> Fremont -> End

Fremont -> Oakland -> San Francisco -> Start
```

# One-way Circular LinkedList

**One-way Circular List:**
It is a chain storage structure of a linear table, which is connected by a node in a unidirectional manner, and is connected to form a ring, and each node is composed of data and a pointer next.

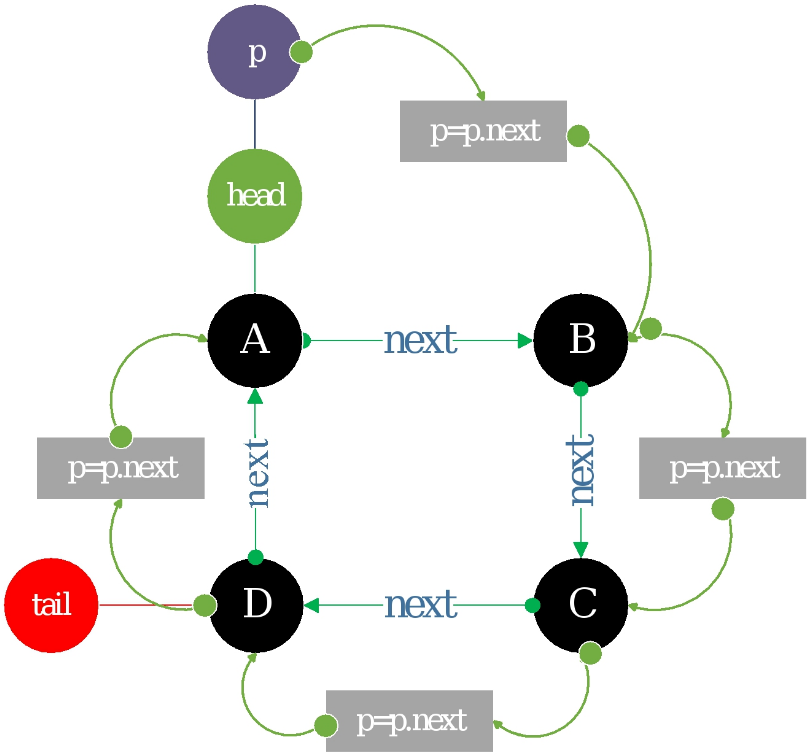

A
next
B
next
next
D
next
C

## UML Diagram


Node
data : String
next : Node
Node(data : String)

```
class Node{
    constructor( data, next){
        this.data = data;
        this.next = next;
    }

    getData(){
        return this.data;
    }
}
```

## 1. One-way Circular Linked List initialization and traversal output .


p
p=p.next
head
A
next
B
p=p.next
next
next
p=p.next
tail
D
next
C
p=p.next

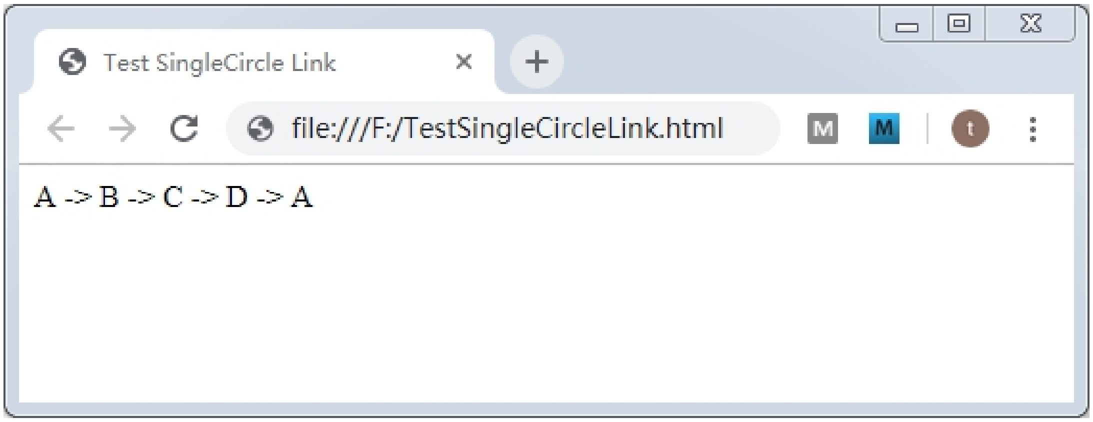
**Create a TestSingleCircleLink.html with Notepad**

```
<script type ="text/javascript" >
    class Node{
        constructor( data, next){
            this.data = data;
            this.next = next;
        }

        getData(){
            return this.data;
        }
    }
```

```
class SingleCircleLink{
    init() {
        // the first node called head node
        this.head = new Node( "A" );
        this.head.next = null ;

        var nodeB = new Node( "B" );
        nodeB.next = null ;
        this.head.next = nodeB;

        var nodeC = new Node( "C" );
        nodeC.next = null ;
        nodeB.next = nodeC;

        // the last node called tail node
        this.tail = new Node( "D" );
        this.tail.next = this.head;
        nodeC.next = this.tail;
    }

    print() {
        var p = this.head;
        do {
            var data = p.getData();
            document.write( data + " -> " );
            p = p.next;
        } while ( p != this.head);

        var data = p.getData();
        document.write( data + "<br><br>" );
    }
}
```

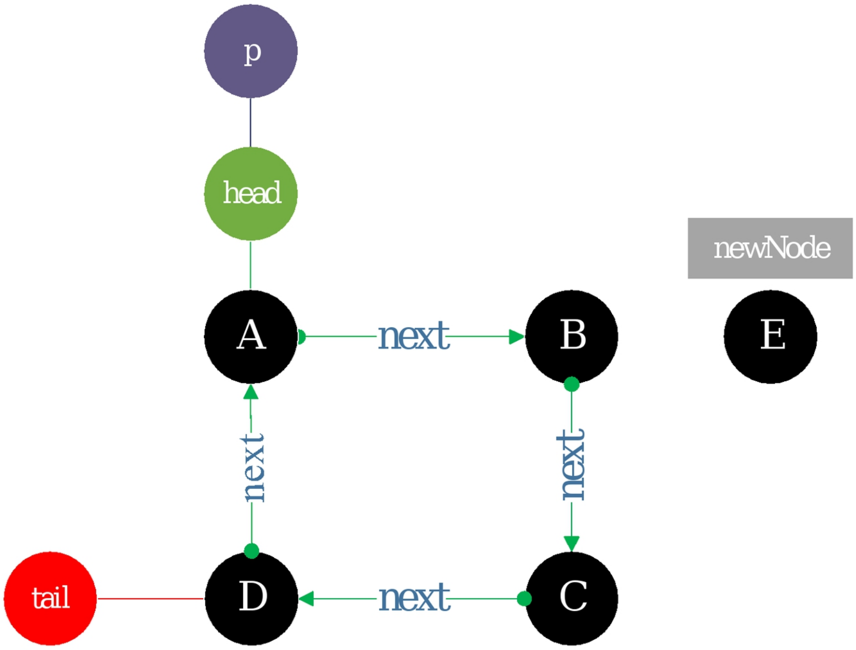
```
    //////////////////////testing////////////////////

    var singleCircleLink = new SingleCircleLink();
    var head = singleCircleLink.init();
    singleCircleLink.print( head);
</script>
```

**Result:**

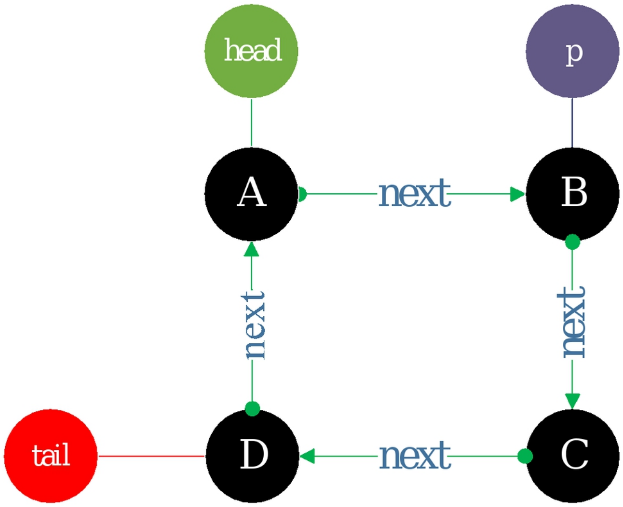

Test SingleCircle Link
file:///F:/TestSingleCircleLink.html

```
A -> B -> C -> D -> A
```

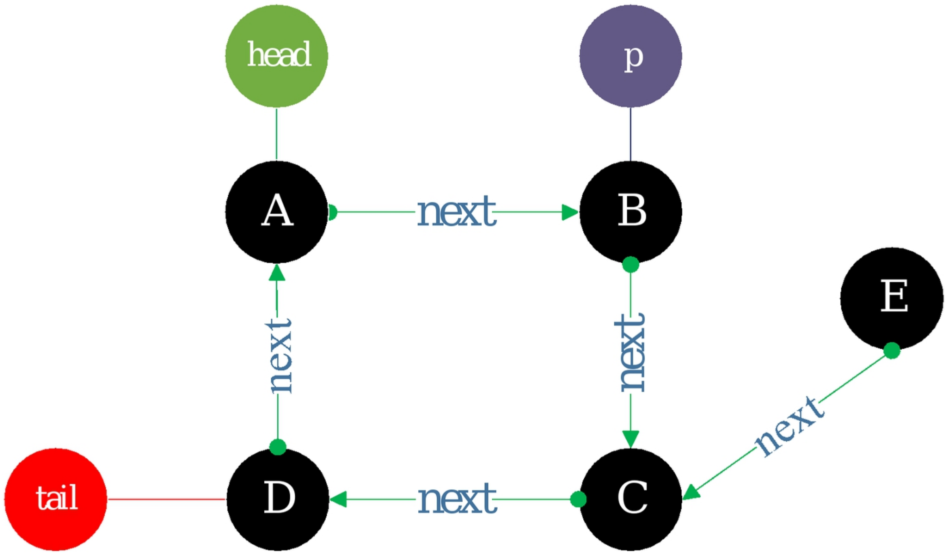
### 3. Insert a node **E** in position 2.


p
head
newNode
A
next
B
E
next
next
tail
D
next
C

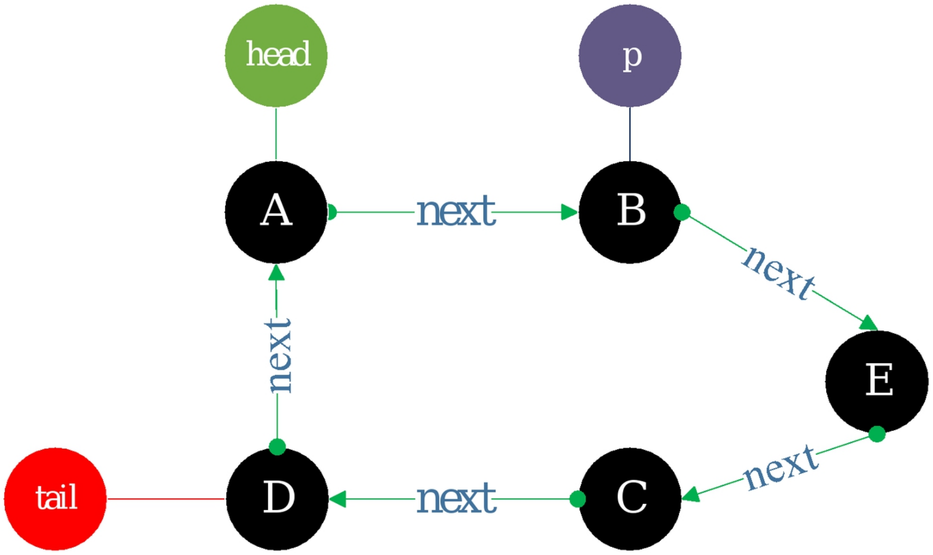
p=p.next


head
p
A
next
B
next
next
tail
D
next
C

newNode.next = p.next


head
p
A
next
B
E
next
next
next
tail
D
next
C

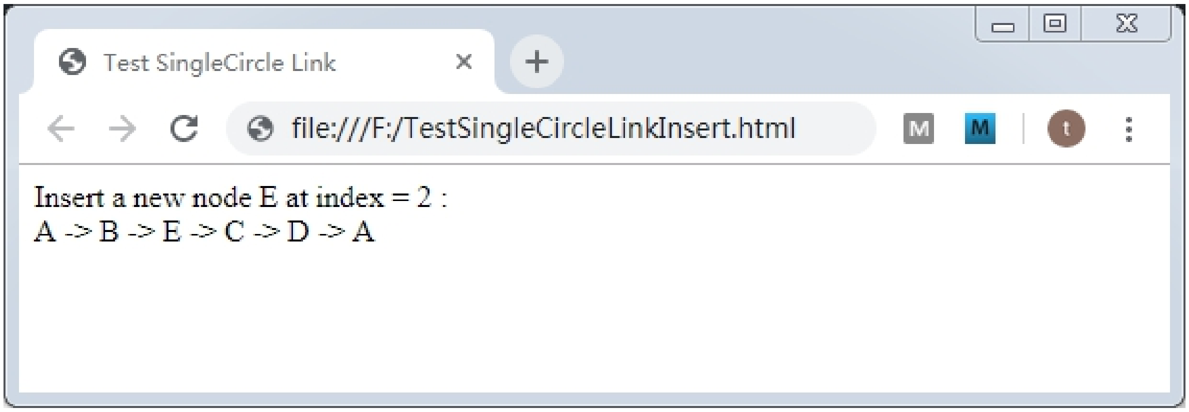
p.next = newNode


head
p
A
next
B
next
next
E
next
tail
D
next
C

**Create a TestSingleCircleLinkInsert.html with Notepad**

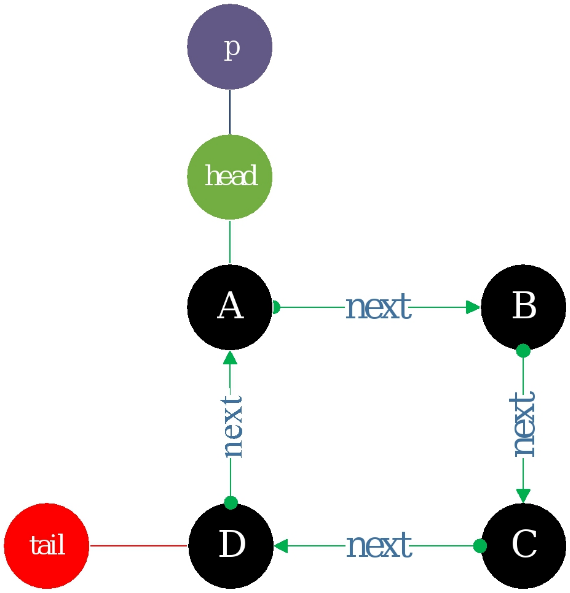
```
<script type ="text/javascript" >
    class Node{
        constructor( data, next){
            this.data = data;
            this.next = next;
        }
        getData(){
            return this.data;
        }
    }

    class SingleCircleLink{
        init() {
            // the first node called head node
            this.head = new Node( "A" );
            this.head.next = null ;

            var nodeB = new Node( "B" );
            nodeB.next = null ;
            this.head.next = nodeB;

            var nodeC = new Node( "C" );
            nodeC.next = null ;
            nodeB.next = nodeC;

            // the last node called tail node
            this.tail = new Node( "D" );
            this.tail.next = this.head;
            nodeC.next = this.tail;
        }

        insert( insertPosition, newNode) {
            var p = this.head;
            var i = 0 ;
            // Move the node to the insertion position
            while ( p.next != null && i < insertPosition - 1 ) {
                p = p.next;
                i++;
            }
            newNode.next = p.next;
            p.next = newNode;
        }

        print() {
            var p = this.head;
            do {
                var data = p.getData();
                document.write( data + " -> " );
                p = p.next;
            } while ( p != this.head);

            var data = p.getData();
            document.write( data + "<br><br>" );
        }
    }

    ////////////////////testing//////////////////

    var singleCircleLink = new SingleCircleLink();
    var head = singleCircleLink.init();

    document.write( "Insert a new node E at index = 2 : <br>" );
    singleCircleLink.insert( 2 , new Node( "E" ));

    singleCircleLink.print( head);
</script>
```

**Result:**

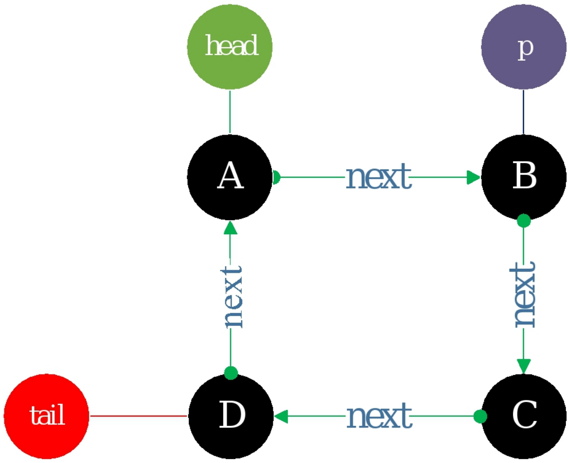

Test SingleCircle Link
file:///F:/TestSingleCircleLinkInsert.html

```
Insert a new node E at index = 2 :
A -> B -> E -> C -> D -> A
```

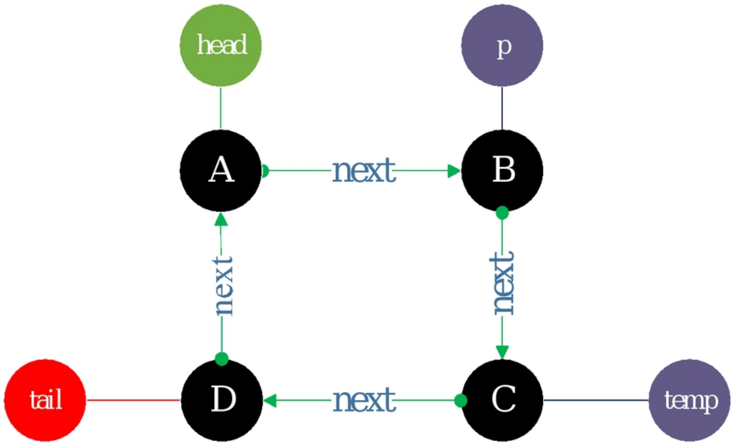
### 4. Delete the index=2 node.


p
head
A
next
B
next
next
tail
D
next
C

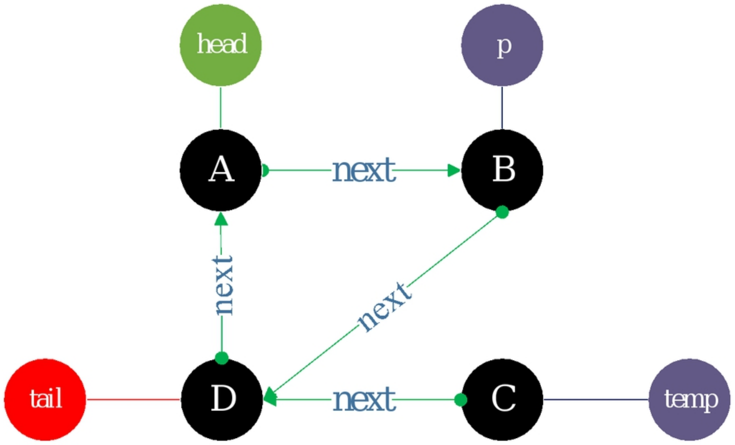
p=p.next


head
p
A
next
B
next
next
tail
D
next
C

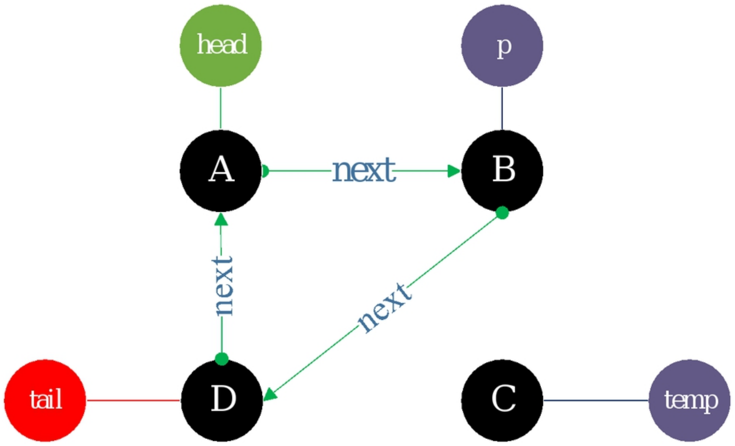
Node temp = p.next


head
p
A
next
B
next
next
tail
D
next
C
temp

p.next = p.next.next


head
p
A
next
B
next
next
tail
D
next
C
temp

temp.next = null

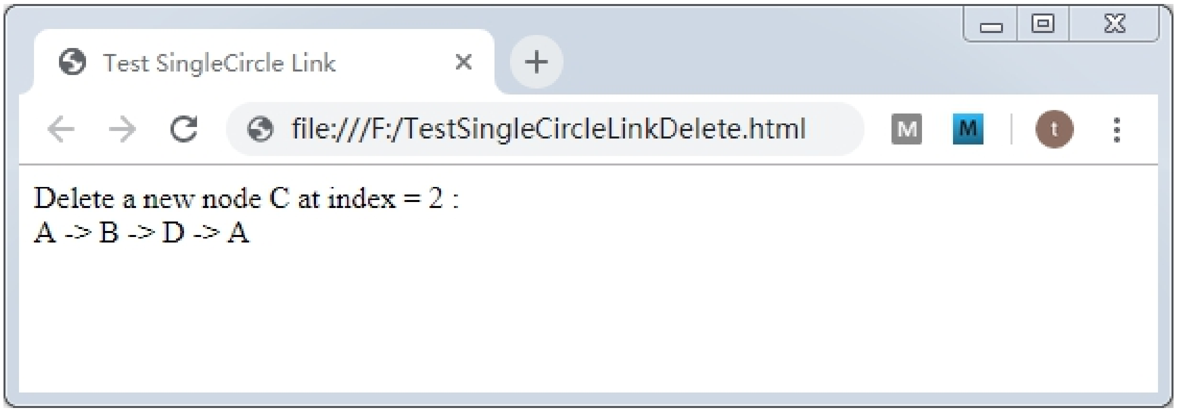

head
p
A
next
B
next
next
tail
D
C
temp

**Create a TestSingleCircleLinkDelete.html with Notepad**

```
<script type ="text/javascript" >
  class Node{
    constructor( data, next){
        this.data = data;
        this.next = next;
    }
    getData(){
        return this.data;
    }
}

class SingleCircleLink{
    init() {
        // the first node called head node
        this.head = new Node( "A" );
        this.head.next = null ;

        var nodeB = new Node( "B" );
        nodeB.next = null ;
        this.head.next = nodeB;

        var nodeC = new Node( "C" );
        nodeC.next = null ;
        nodeB.next = nodeC;

        // the last node called tail node
        this.tail = new Node( "D" );
        this.tail.next = this.head;
        nodeC.next = this.tail;
    }

    remove( removePosition) {
        var p = this.head;
        var i = 0 ;
        while ( p.next != null && i < removePosition - 1 ) {
            p = p.next;
            i++;
        }
        var temp = p.next;
```

```
            p.next = p.next.next;
            temp.next = null ;
        }

        print() {
            var p = this.head;
            do {
                var data = p.getData();
                document.write( data + " -> " );
                p = p.next;
            } while ( p != this.head);

            var data = p.getData();
            document.write( data + "<br><br>" );
        }
    }

    ////////////////////testing//////////////////

    var singleCircleLink = new SingleCircleLink();
    singleCircleLink.init();

    document.write( "Delete a new node C at index = 2 : <br>" );
    singleCircleLink.remove( 2 );

    singleCircleLink.print();
</script>
```

**Result:**

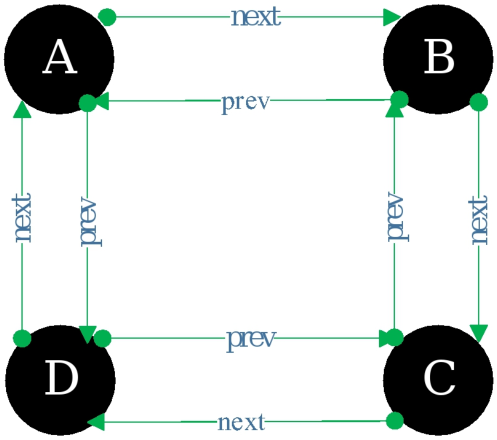

Test SingleCircle Link
file:///F:/TestSingleCircleLinkDelete.html

```
Delete a new node C at index = 2 :
A -> B -> D -> A
```

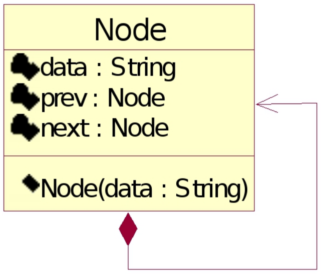
# Two-way Circular LinkedList

**Two-way Circular List:**

It is a chain storage structure of a linear table. The nodes are connected in series by two directions, and is connected to form a ring. Each node is composed of data , pointing to the previous node prev and pointing to the next node next .

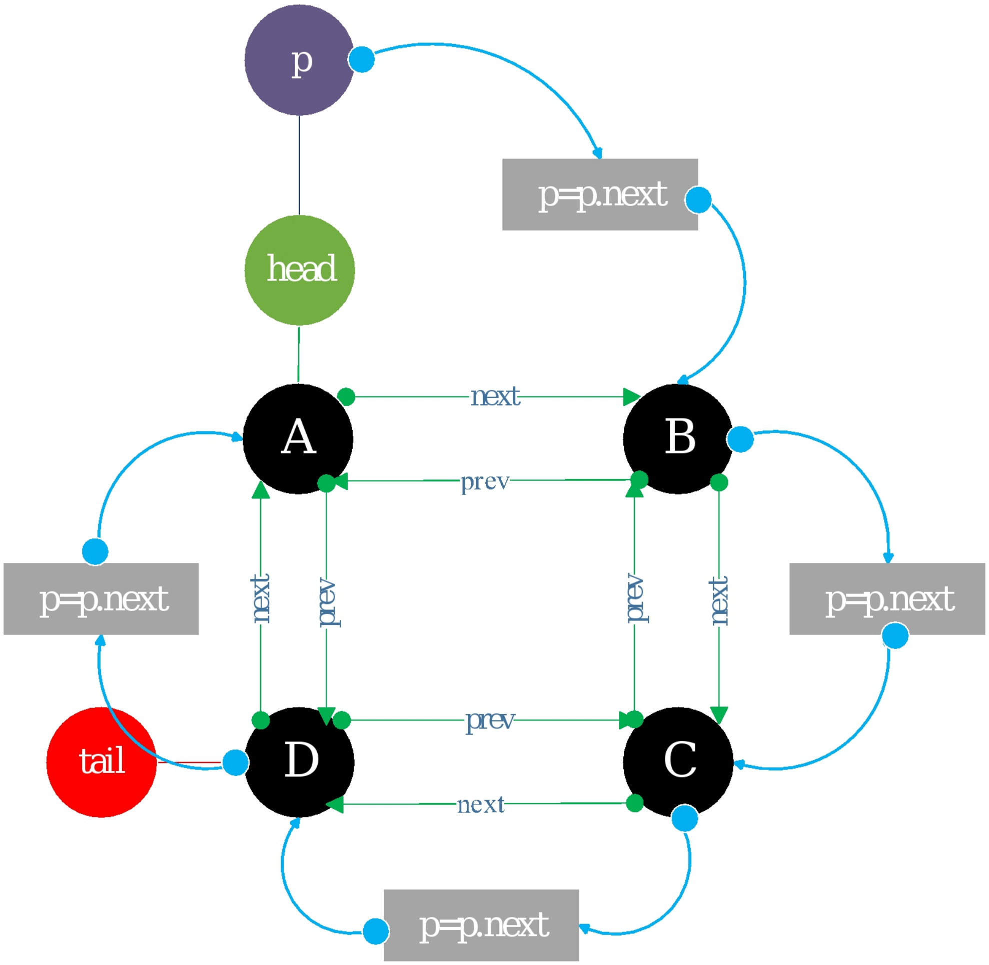

A
B
C
D
next
prev
next
prev
prev
next
prev
next

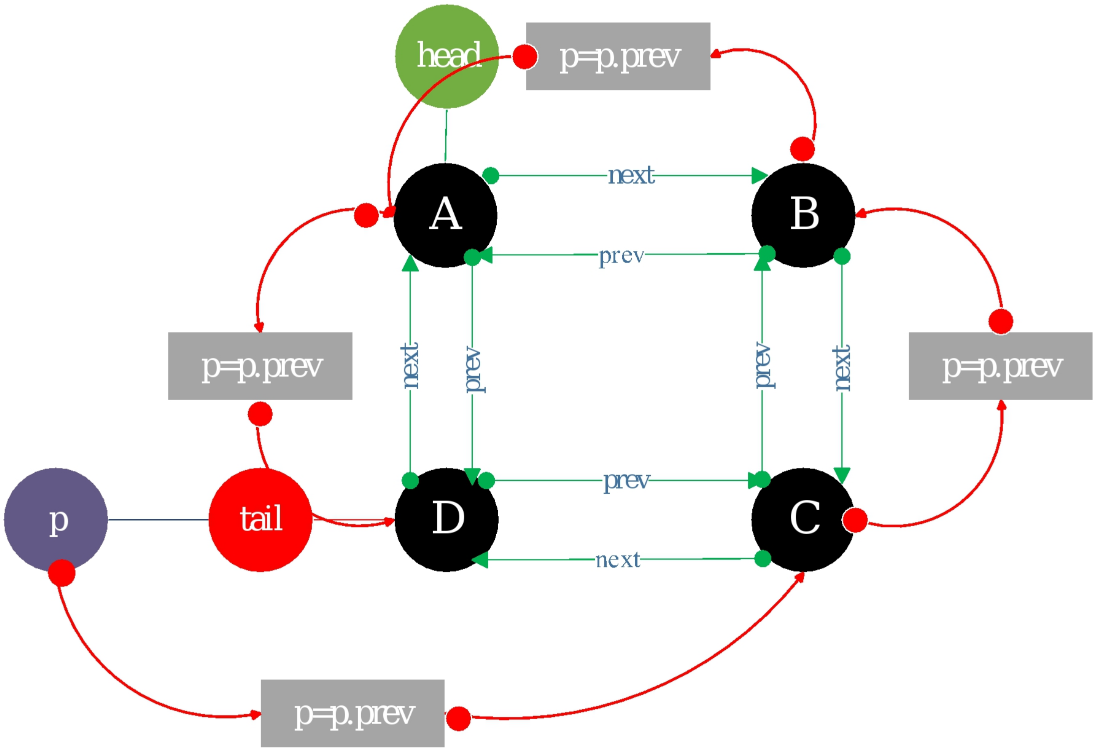
## UML Diagram


Node
data : String
prev : Node
next : Node
Node(data : String)

```
class Node{
    constructor( data, prev, next){
        this.data = data;
        this.prev = prev;
        this.next = next;
    }

    getData(){
        return this.data;
    }
}
```

## 1. Two-way Circular Linked List initialization and traversal output .

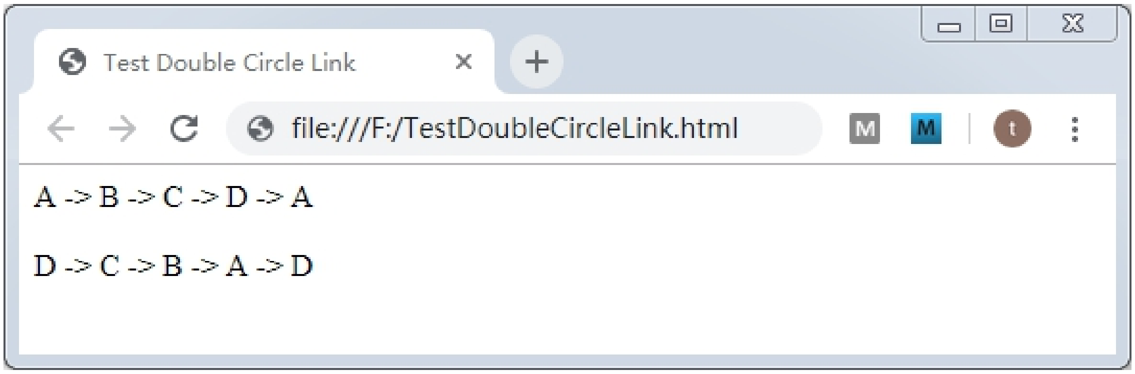

p
p=p.next
head
A
next
B
prev
p=p.next
next
prev
prev
next
p=p.next
tail
D
prev
C
next
p=p.next


head
p=p.prev
A
next
B
prev
p=p.prev
next
prev
prev
next
p=p.prev
p
tail
D
prev
C
next
p=p.prev

**Create a TestDoubleCircleLink.html with Notepad**

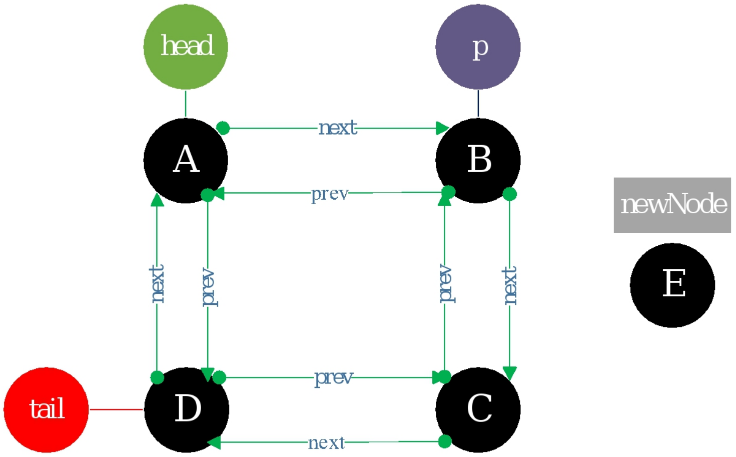
```
<script type ="text/javascript" >
    class Node{
        constructor( data, prev, next){
            this.data = data;
            this.prev = prev;
            this.next = next;
        }

        getData(){
            return this.data;
        }
    }

    class DoubleCircleLink{
        init() {
            // the first node called head node
            this.head = new Node( "A" );
            this.head.next = null ;
            this.head.prev = null ;

            var nodeB = new Node( "B" );
            nodeB.next = null ;
            nodeB.prev = this.head;
            this.head.next = nodeB;

            var nodeC = new Node( "C" );
            nodeC.next = null ;
            nodeC.prev = nodeB;
            nodeB.next = nodeC;

            // the last node called tail node
            this.tail = new Node( "D" );
            this.tail.next = this.head;
            this.tail.prev = nodeC;
        nodeC.next = this.tail;
        this.head.prev = this.tail;
    }






    print() {
        var p = this.head;
        do {
            var data = p.getData();
            document.write( data + " -> " );
            p = p.next;
        } while ( p != this.head);

        var data = p.getData();
        document.write( data + "<br><br>" );

        p = this.tail;
        do {
            data = p.getData();
            document.write( data + " -> " );
            p = p.prev;
        } while ( p != this.tail);

        data = p.getData();
        document.write( data + "<br><br>" );
    }
}

////////////////////testing/////////////////
var doubleCircleLink = new DoubleCircleLink();
doubleCircleLink.init();
doubleCircleLink.print();
</script>
```

**Result:**

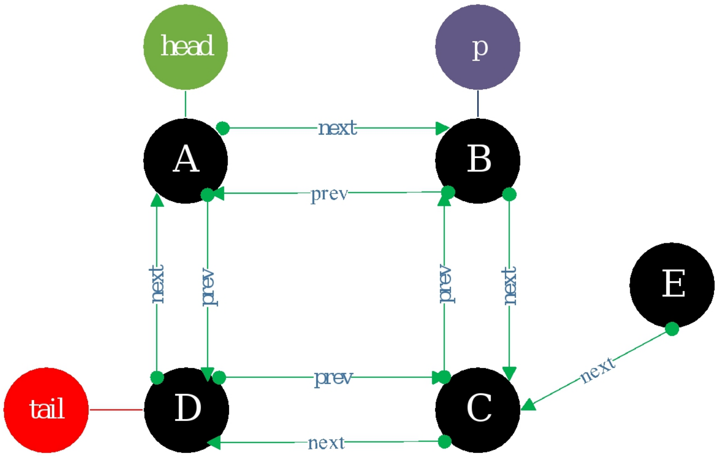

Test Double Circle Link
file:///F:/TestDoubleCircleLink.html

```
A -> B -> C -> D -> A

D -> C -> B -> A -> D
```

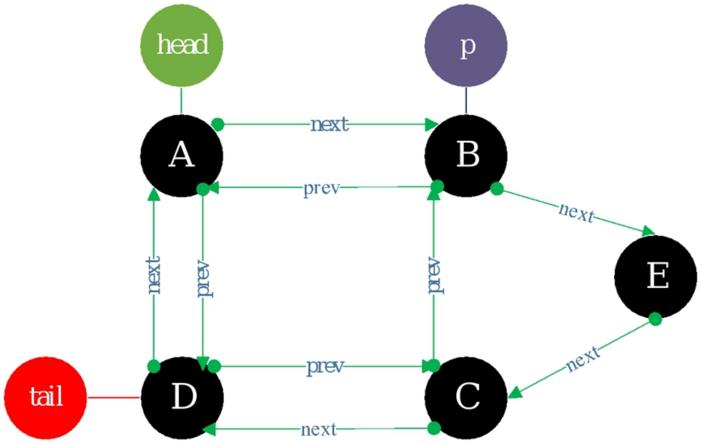
### 3. Insert a node E in position 2.


head
p
A
next
B
prev
newNode
E
next
prev
prev
next
tail
D
prev
C
next

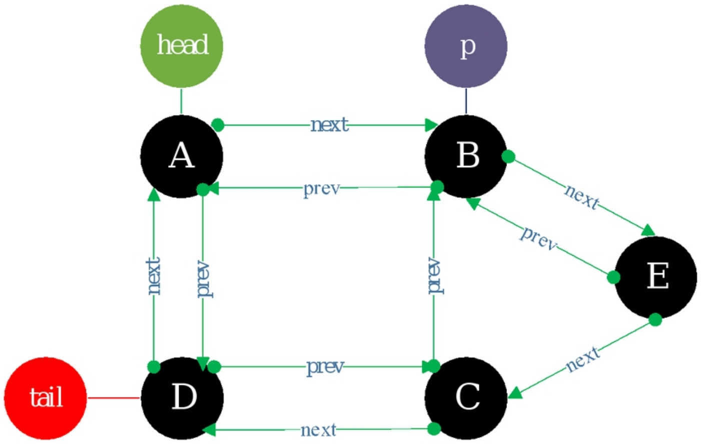
newNode.next = p.next


head
p
A
next
B
prev
next
prev
prev
next
E
next
tail
D
prev
C
next

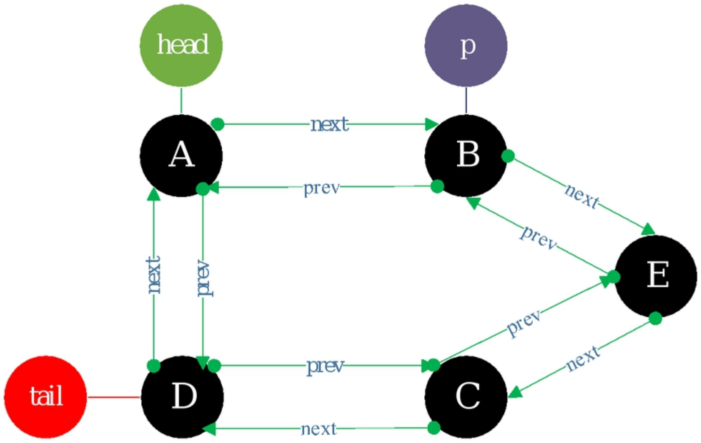
p.next =newNode


head
p
A
B
E
D
C
tail
next
prev
next
next
prev
prev
prev
next
next

newNode.prev =p


head
p
A
B
E
D
C
tail
next
prev
next
prev
next
prev
prev
prev
next
next

newNode.next.prev =newNode


head
p
A
B
E
D
C
tail
next
prev
next
prev
next
prev
prev
prev
next
next

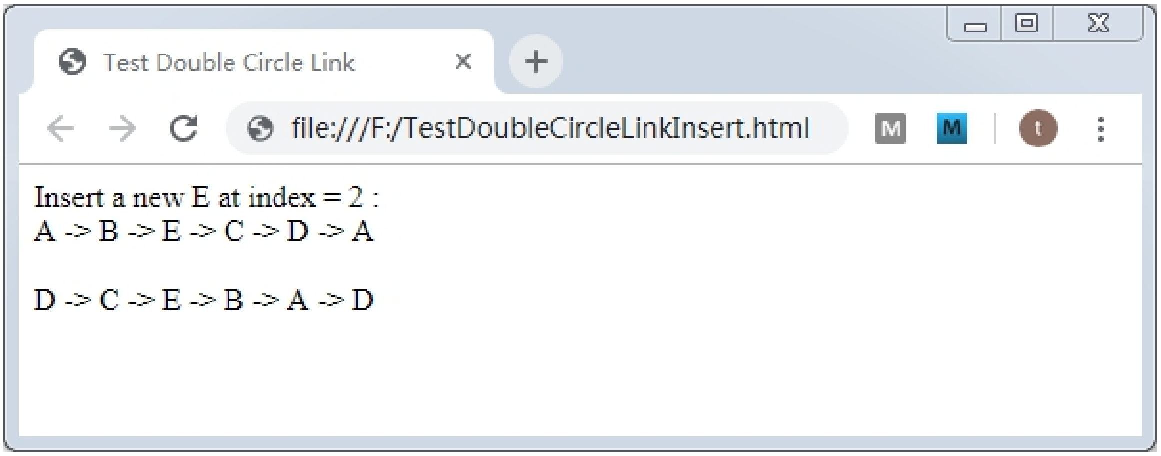
**Create a TestDoubleCircleLinkInsert.html with Notepad**

```
<script type ="text/javascript" >
    class Node{
        constructor( data, prev, next){
            this.data = data;
            this.prev = prev;
            this.next = next;
        }

        getData(){
            return this.data;
        }
    }

    class DoubleCircleLink{
        init() {
            // the first node called head node
            this.head = new Node( "A" );
            this.head.next = null ;
            this.head.prev = null ;

            var nodeB = new Node( "B" );
            nodeB.next = null ;
            nodeB.prev = this.head;
            this.head.next = nodeB;

            var nodeC = new Node( "C" );
            nodeC.next = null ;
            nodeC.prev = nodeB;
            nodeB.next = nodeC;

            // the last node called tail node
            this.tail = new Node( "D" );
            this.tail.next = this.head;
            this.tail.prev = nodeC;
            nodeC.next = this.tail;
            this.head.prev = this.tail;
        }
```

```
insert( insertPosition, newNode) {
    var p = this.head;
    var i = 0 ;
    //Move the node to the insertion position
    while ( p.next != null && i < insertPosition - 1 ) {
        p = p.next;
        i++;
    }

    newNode.next = p.next;
    p.next = newNode;
    newNode.prev = p;
    newNode.next.prev = newNode;
}

print() {
    var p = this.head;
    do {
        var data = p.getData();
        document.write( data + " -> " );
        p = p.next;
    } while ( p != this.head);

    var data = p.getData();
    document.write( data + "<br><br>" );

    p = this.tail;
    do {
        data = p.getData();
        document.write( data + " -> " );
        p = p.prev;
    } while ( p != this.tail);
```

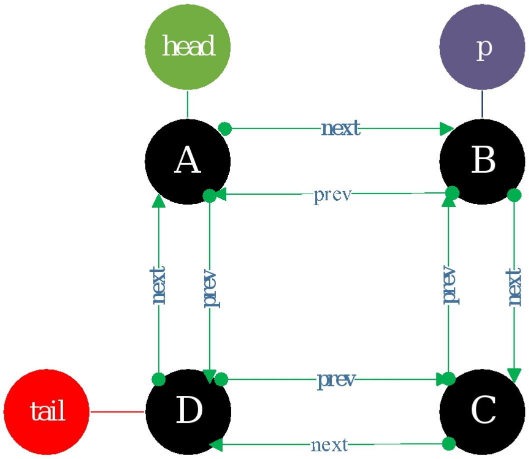
```
            data = p.getData();
            document.write( data + "<br><br>" );
        }
    }







    ////////////////////testing/////////////////

    var doubleCircleLink = new DoubleCircleLink();
    doubleCircleLink.init();

    document.write( "Insert a new E at index = 2 : <br>" );
    doubleCircleLink.insert( 2 , new Node( "E" ));

    doubleCircleLink.print();
</script>
```

**Result:**

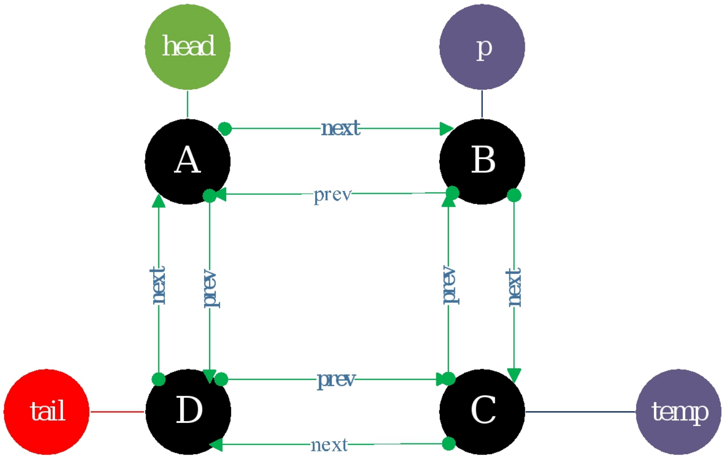

Test Double Circle Link
file:///F:/TestDoubleCircleLinkInsert.html

```
Insert a new E at index = 2 :
A -> B -> E -> C -> D -> A

D -> C -> E -> B -> A -> D
```

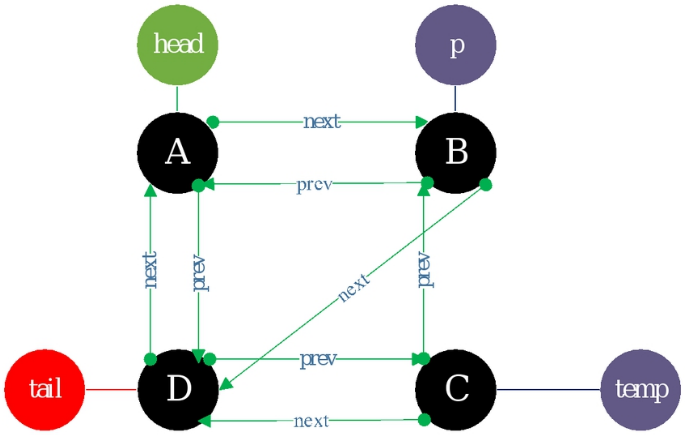
## 4. Delete the index=2 node.


head
p
next
A
B
prev
next
prev
prev
next
tail
D
prev
C
next

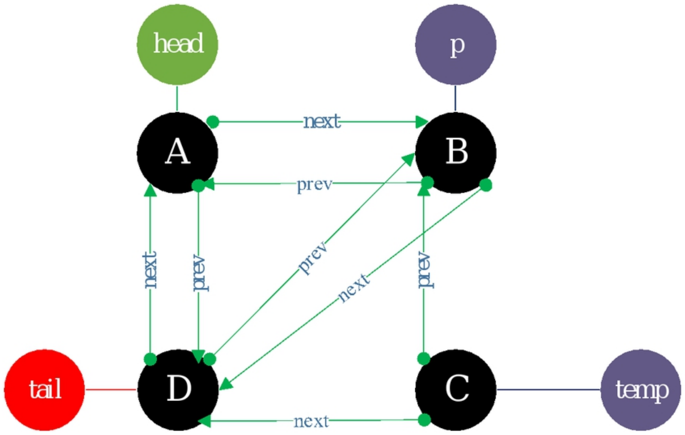
Node temp = p.next


head
p
next
A
B
prev
next
prev
prev
next
tail
D
prev
C
temp
next

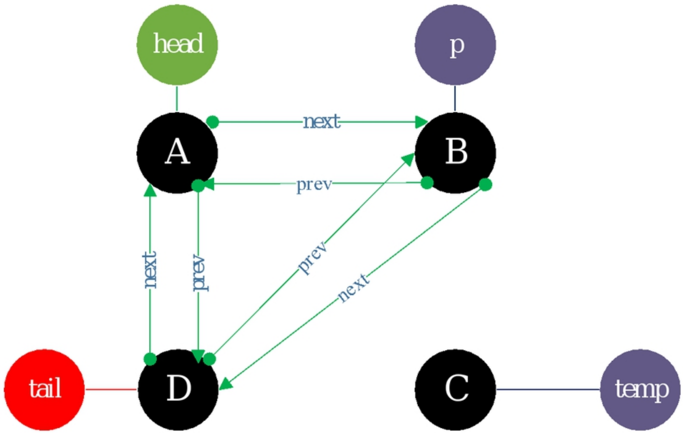
p.next = p.next.next


head
p
A
B
next
prev
next
prev
next
prev
tail
D
C
temp
prev
next

p.next.prev = p


head
p
A
B
next
prev
next
prev
prev
next
prev
tail
D
C
temp
next

temp.next = null
temp.prev = null


head
p
A
B
next
prev
next
prev
prev
next
tail
D
C
temp

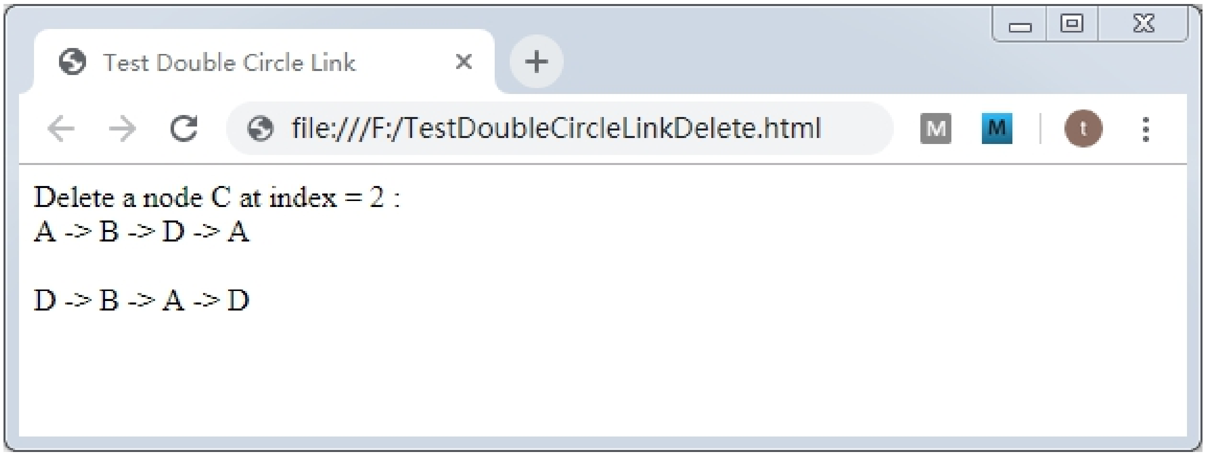
**Create a TestDoubleCircleLinkDelete.html with Notepad**

```
<script type ="text/javascript" >
    class Node{
        constructor( data, prev, next){
            this.data = data;
            this.prev = prev;
            this.next = next;
        }

        getData(){
            return this.data;
        }
    }

    class DoubleCircleLink{
        init() {
            // the first node called head node
            this.head = new Node( "A" );
            this.head.next = null ;
            this.head.prev = null ;

            var nodeB = new Node( "B" );
            nodeB.next = null ;
            nodeB.prev = this.head;
            this.head.next = nodeB;

            var nodeC = new Node( "C" );
            nodeC.next = null ;
            nodeC.prev = nodeB;
            nodeB.next = nodeC;

            // the last node called tail node
            this.tail = new Node( "D" );
            this.tail.next = this.head;
            this.tail.prev = nodeC;
            nodeC.next = this.tail;
            this.head.prev = this.tail;
        }
```

```
remove( removePosition) {
    var p = this.head;
    var i = 0 ;
    while ( p.next != null && i < removePosition - 1 ) {
        p = p.next;
        i++;
    }

    var temp = p.next;
    p.next = p.next.next;
    p.next.prev = p;
    temp.next = null ; //Set the delete node next to null
    temp.prev = null ; // Set the delete node prev to null

}

print() {
    var p = this.head;
    do {
        var data = p.getData();
        document.write( data + " -> " );
        p = p.next;
    } while ( p != this.head);

    var data = p.getData();
    document.write( data + "<br><br>" );

    p = this.tail;
    do {
        data = p.getData();
        document.write( data + " -> " );
        p = p.prev;
```

```
            } while ( p != this.tail);

            data = p.getData();
            document.write( data + "<br><br>" );
        }
    }







    ////////////////////testing//////////////////

    var doubleCircleLink = new DoubleCircleLink();
    doubleCircleLink.init();

    document.write( "Delete a node C at index = 2 : <br>" );
    doubleCircleLink.remove( 2 );

    doubleCircleLink.print();
</script>
```

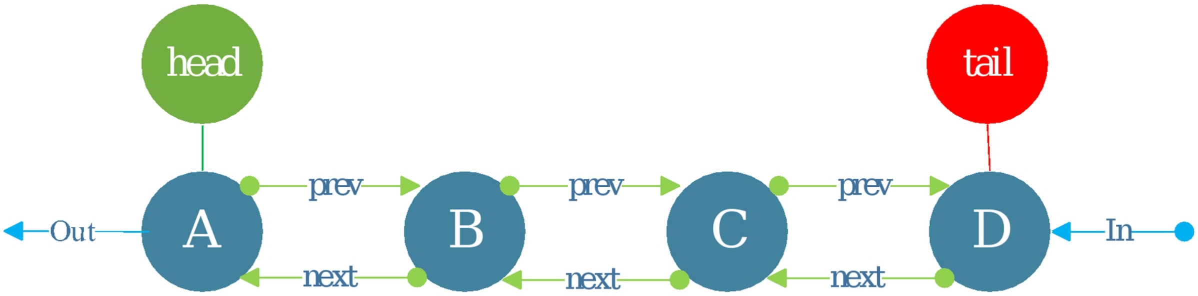
**Result:**


Test Double Circle Link
file:///F:/TestDoubleCircleLinkDelete.html

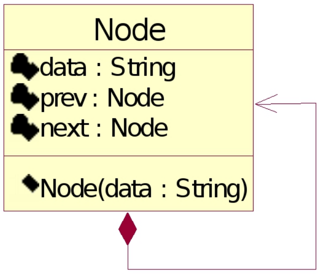
```
Delete a node C at index = 2 :
A -> B -> D -> A

D -> B -> A -> D
```

# Queue

**Queue:**
FIFO (First In First Out) sequence.


head
tail
prev
prev
prev
Out
A
B
C
D
In
next
next
next

**UML Diagram**

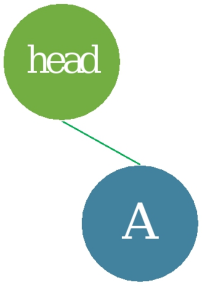

Node
data : String
prev : Node
next : Node
Node(data : String)

```
class Node{
    constructor( data, prev, next){
        this.data = data;
        this.prev = prev;
        this.next = next;
    }

    getData(){
        return this.data;
    }
}
```

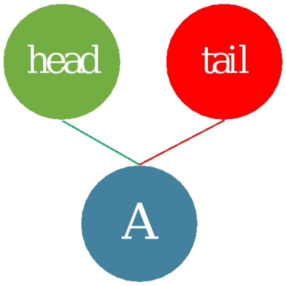
## 1. Queue initialization and traversal output .

### Initialization Insert A

```
head = new Node(" A" );
```

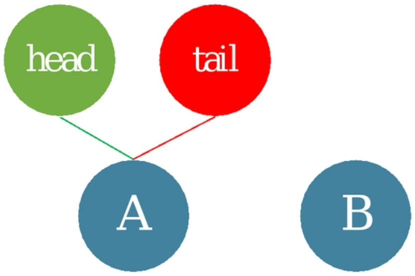


```
tail = head;
```

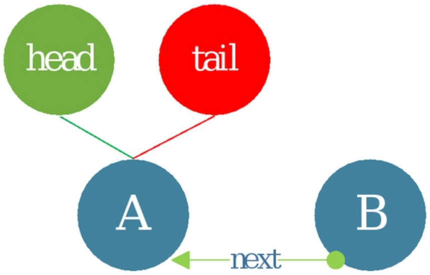


## Initialization Insert B

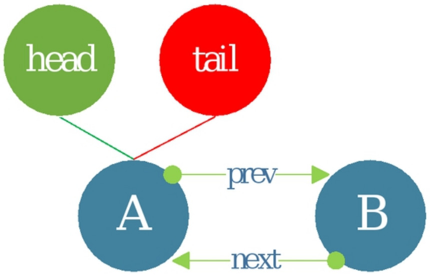
```
newNode = new Node(" B" );
```


head
tail
A
B

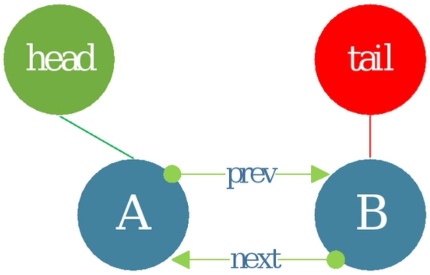
```
newNode.next = tail;
```


head
tail
A
B
next

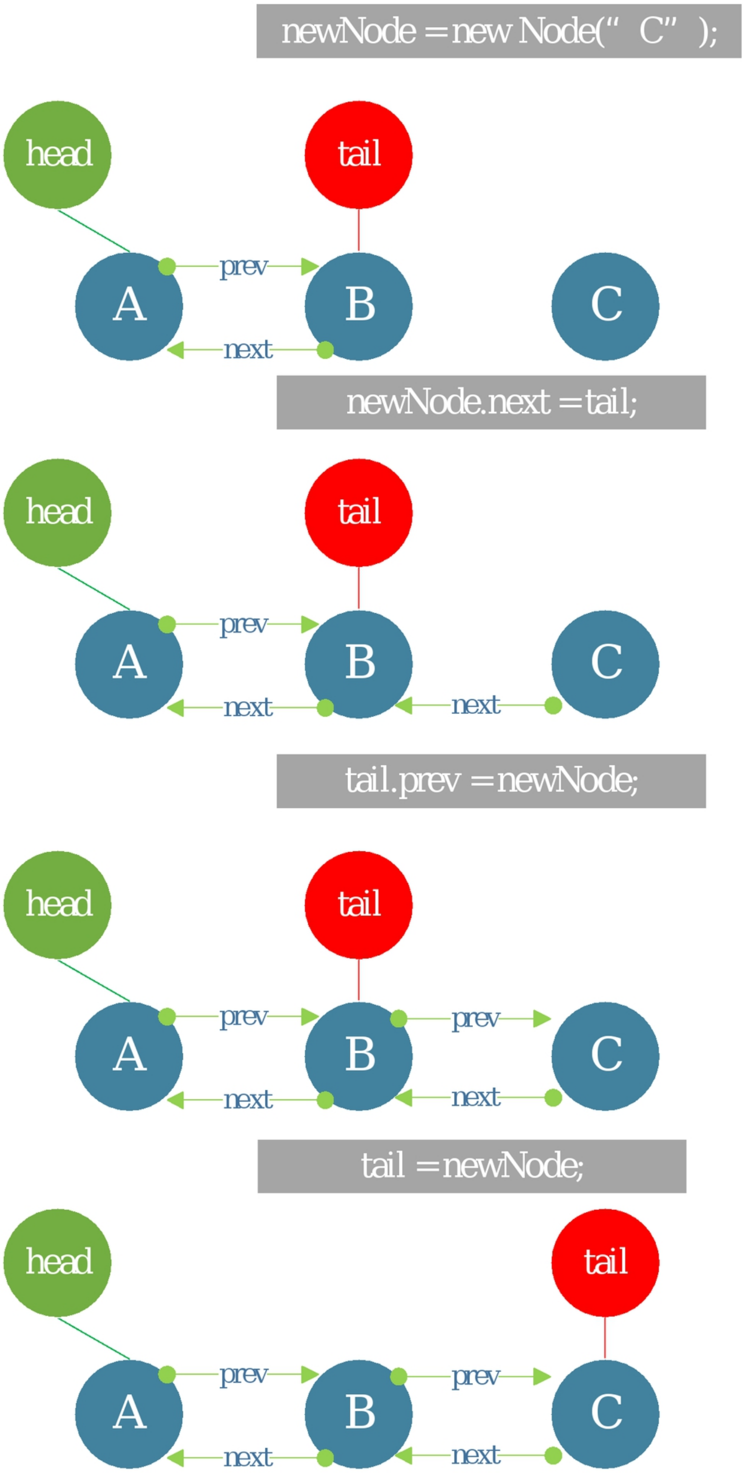
```
tail.prev = newNode;
```


head
tail
prev
A
B
next

```
tail = newNode;
```


head
tail
prev
A
B
next

## Initialization Insert C

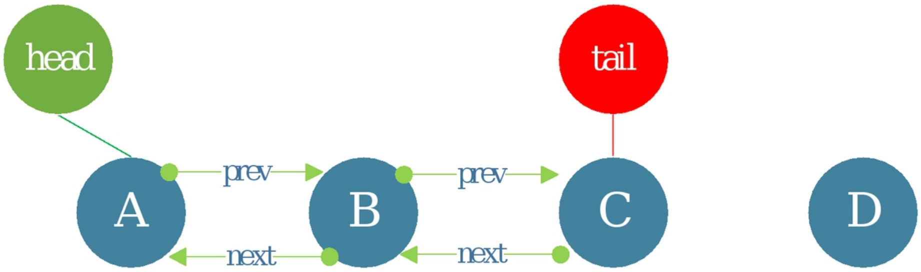


```
newNode =new Node(“ C” );
newNode.next =tail;
tail.prev =newNode;
tail =newNode;
```

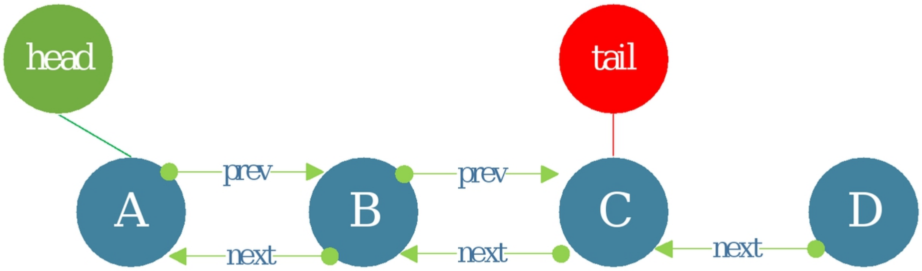
head
tail
A
B
C
prev
next

### **Initialization Insert D**

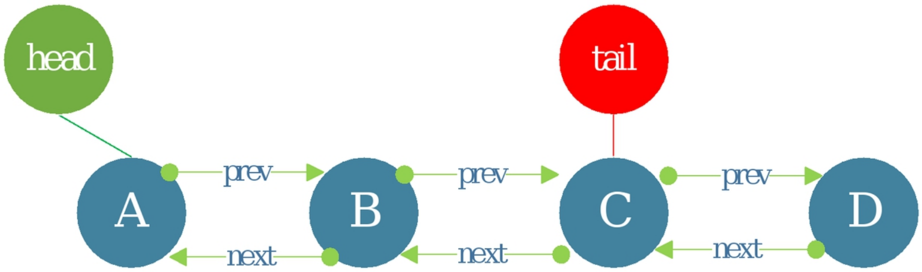
newNode = new Node(" D" );


head
tail
A
B
C
D
prev
prev
next
next

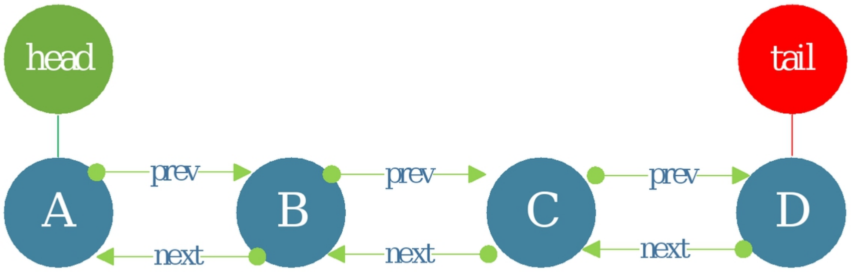
newNode.next = tail;


head
tail
A
B
C
D
prev
prev
next
next
next

tail.prev = newNode;


head
tail
A
B
C
D
prev
prev
prev
next
next
next

tail = newNode;


head
tail
A
B
C
D
prev
prev
prev
next
next
next

### Create a TestQueue.html with Notepad

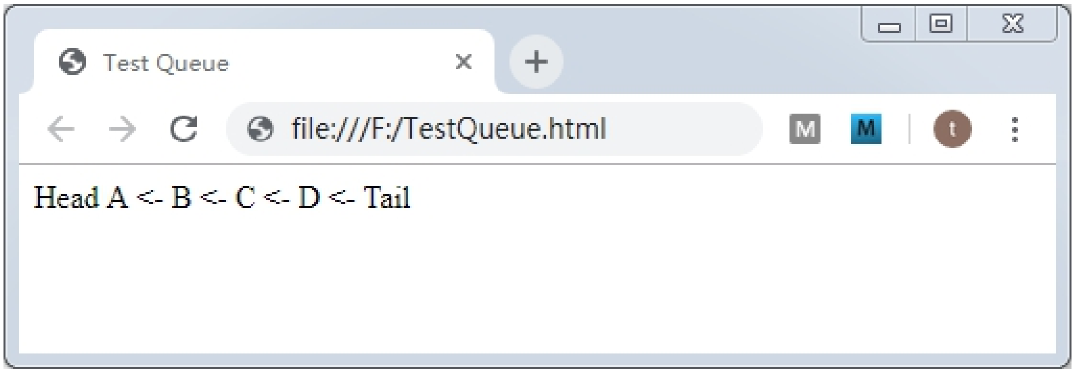
```
<script type ="text/javascript" >
```

```
class Node{
    constructor( data, prev, next){
        this.data = data;
        this.prev = prev;
        this.next = next;
    }

    getData(){
        return this.data;
    }
}
```

```
class Queue{
    constructor(){
        this.head = null ;
        this.tail = null ;
        this.size = 0 ;
    }

    offer( element) {
        if ( this.head == null ) {
            this.head = new Node( element);
            this.tail = this.head;
        } else {
            var newNode = new Node( element);
            newNode.next = this.tail;
            this.tail.prev = newNode;
            this.tail = newNode;
        }
        this.size++;
    }

    poll() {
        var p = this.head;

        if ( p == null ) {
            return null ;
        }

        this.head = this.head.prev;

        p.next = null ;
        p.prev = null ;

        this.size--;
```

```
            return p;
        }

        size() {
            return this.size;
        }
    }




    ////////////////////testing//////////////////

    function print( queue) {
        document.write( "Head " );
        var node = null ;
        while (( node = queue.poll())!= null ) {
            document.write( node.getData() + " <- " );
        }
        document.write( "Tail <br>" );
    }

    var queue = new Queue();
    queue.offer( "A" );
    queue.offer( "B" );
    queue.offer( "C" );
    queue.offer( "D" );

    print( queue);

</script>
```

**Result:**

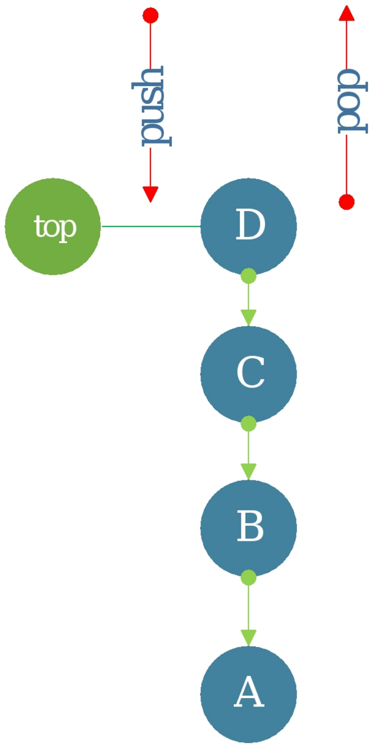

Test Queue
file:///F:/TestQueue.html

```
Head A <- B <- C <- D <- Tail
```

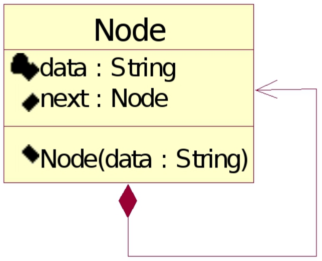
# Stack

**Stack:**

FILO (First In Last Out) sequence.


push
pop
top
D
C
B
A

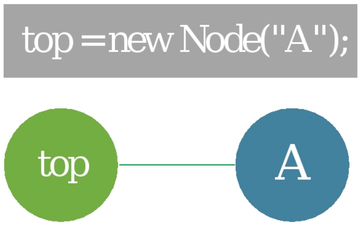
## UML Diagram


Node
data : String
next : Node
Node(data : String)

```
class Node{
    constructor( data, next){
        this.data = data;
        this.next = next;
    }

    getData(){
        return this.data;
    }
}
```

### 1. Stack initialization and traversal output .

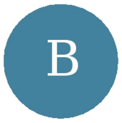
**Push A into Stack**



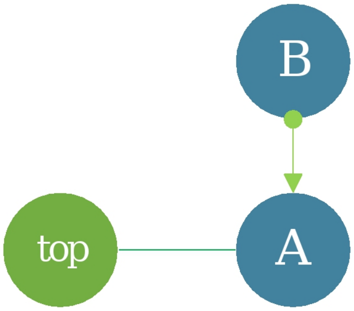
```
top =new Node("A");
```

## Push B into Stack

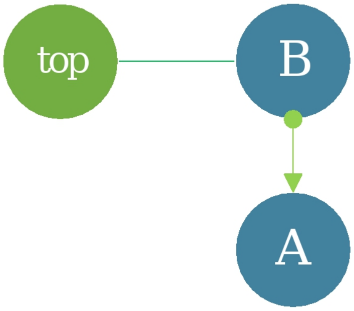
```
newNode = new Node("B");
```



```
newNode.next = top;
```

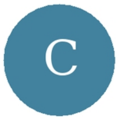


```
top = newNode;
```

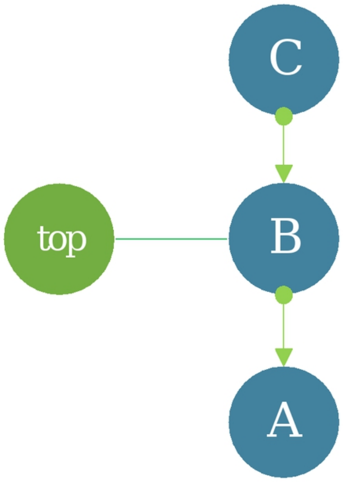


## Push C into Stack

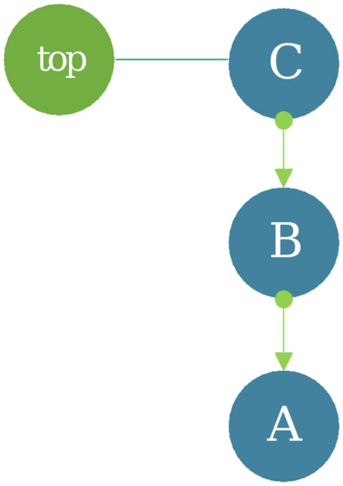
```
newNode = new Node("C");
```



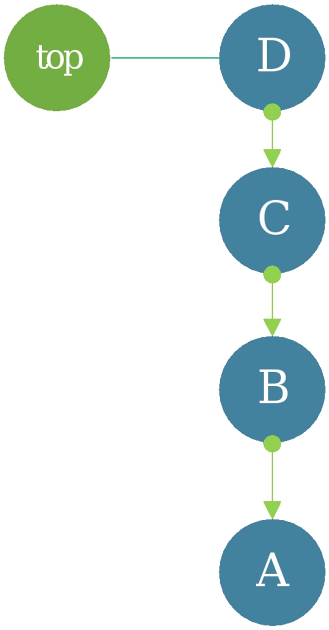
```
newNode.next = top;
```



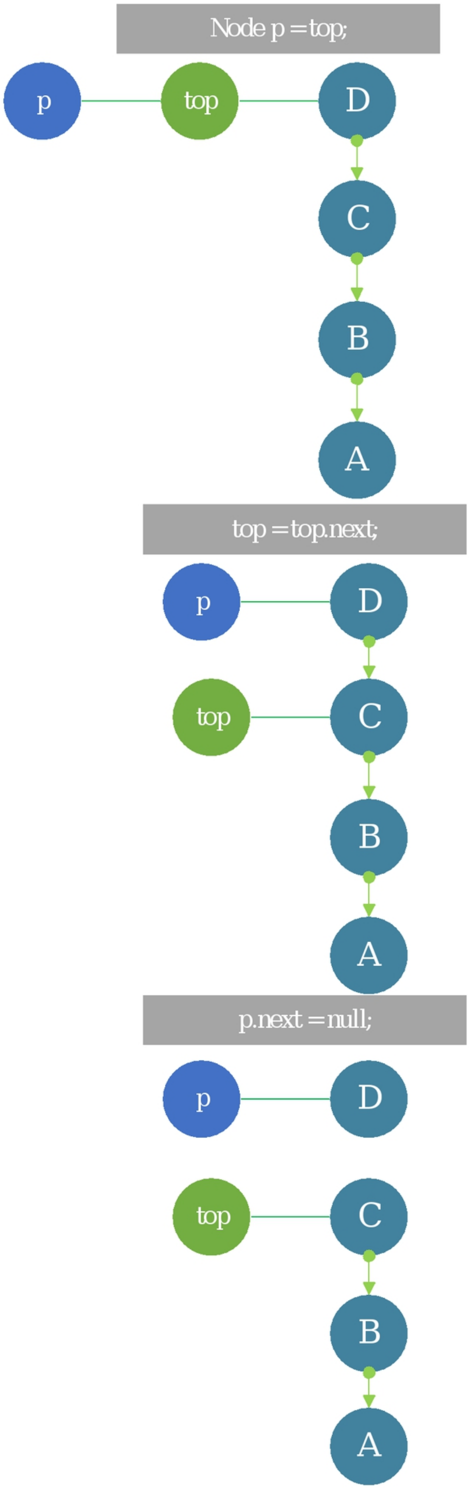
```
top = newNode;
```



**Push D into Stack the same:**



## If pop **D** from Stack:


Node p = top;
p
top
D
C
B
A
top = top.next;
p
D
top
C
B
A
p.next = null;
p
D
top
C
B
A

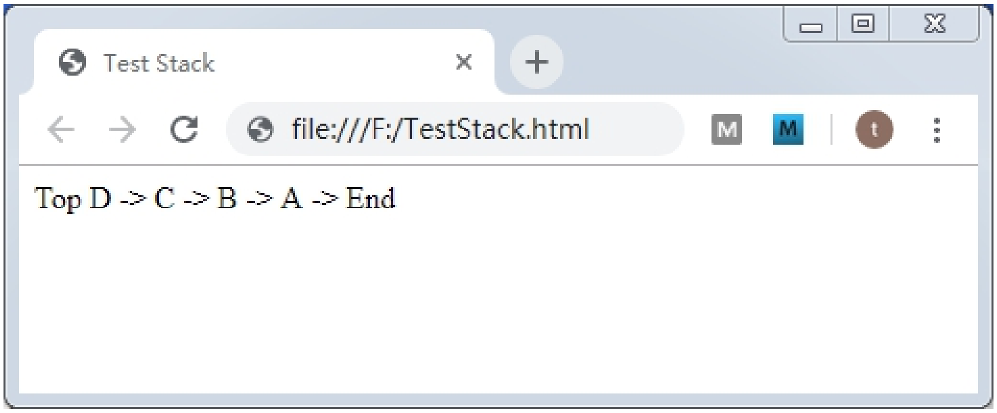
## Create a TestStack.html with Notepad

```
<script type ="text/javascript" >
```

```
class Node{
    constructor( data, next){
        this.data = data;
        this.next = next;
    }
    getData(){
        return this.data;
    }
}

class Stack {
    constructor(){
        this.top = null ;
        this.size = 0 ;
    }

    push( element) {
        if ( this.top == null ) {
            this.top = new Node( element);
        } else {
            var newNode = new Node( element);
            newNode.next = this.top;
            this.top = newNode;
        }
        this.size++;
    }

    pop() {
        if ( this.top == null ){
            return null ;
        }

        var p = this.top;
        this.top = this.top.next; // top move down
        p.next = null ;

        this.size--;
```

```
            return p;
        }


        size() {
            return this.size;
        }

    }

    ////////////////////testing//////////////////

    function print( stack) {
        document.write( "Top " );
        var node = null ;
        while (( node = stack.pop())!= null ) {
            document.write( node.getData() + " -> " );
        }
        document.write( "End <br>" );
    }


    var stack = new Stack();
    stack.push( "A" );
    stack.push( "B" );
    stack.push( "C" );
    stack.push( "D" );
    print( stack);

</script>
```

**Result:**



```
Top D -> C -> B -> A -> End
```

# Recursive Algorithm

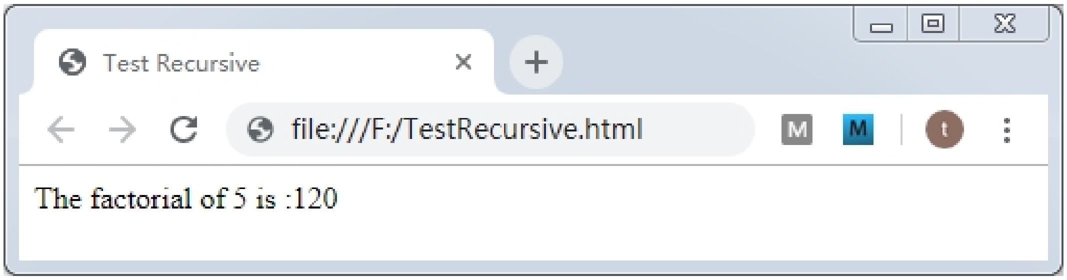
**Recursive Algorithm:**
The program function itself calls its own layer to progress until it reaches a certain condition and step by step returns to the end..

**1. Factorial of n : n*(n-1)*(n-2) ...... *2*1**

**Create a TestRecursive.html with Notepad**

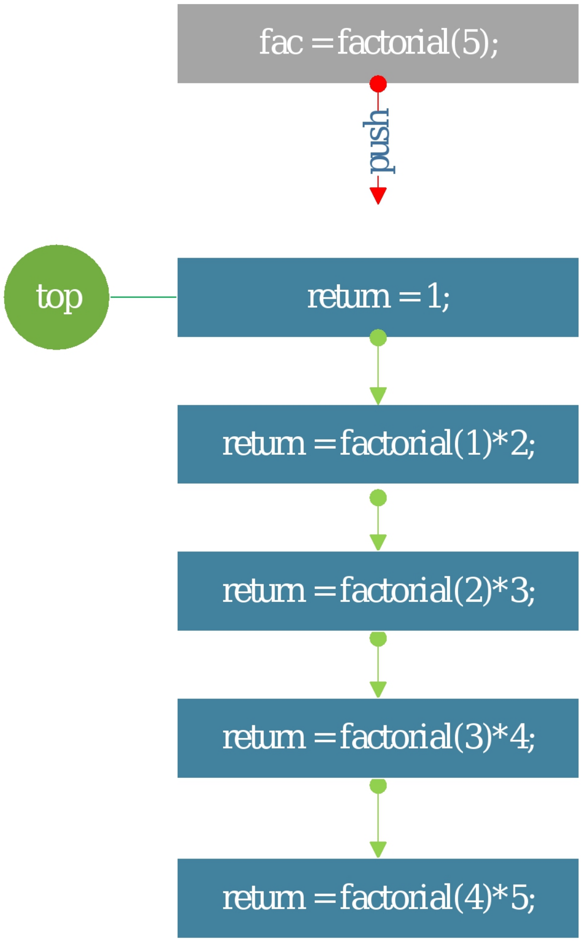
```
<script type ="text/javascript" >
  function factorial( n) {
     if ( n == 1 ) {
        return 1 ;
     } else {
        //Recursively call yourself until the end of the return
        return factorial( n - 1 ) * n;
     }
  }

  ////////////////////testing/////////////////

  var n = 5 ;
  var fac = factorial( n);
  document.write( "The factorial of 5 is :" + fac);
```

**Result:**

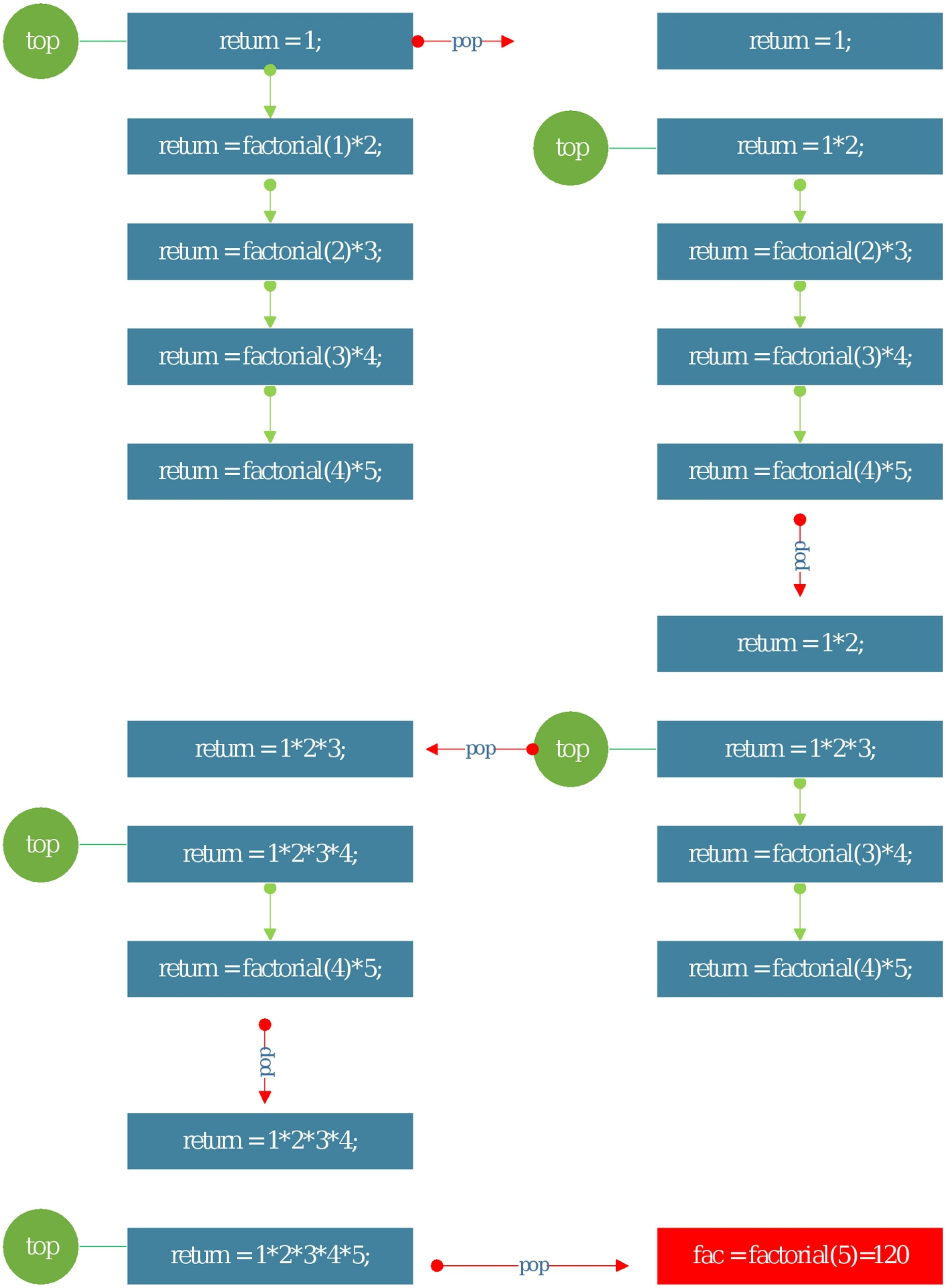

Test Recursive
file:///F:/TestRecursive.html

```
The factorial of 5 is :120
```

**Graphical analysis:**



fac =factorial(5);
push
top
return =1;
return =factorial(1)*2;
return =factorial(2)*3;
return =factorial(3)*4;
return =factorial(4)*5;

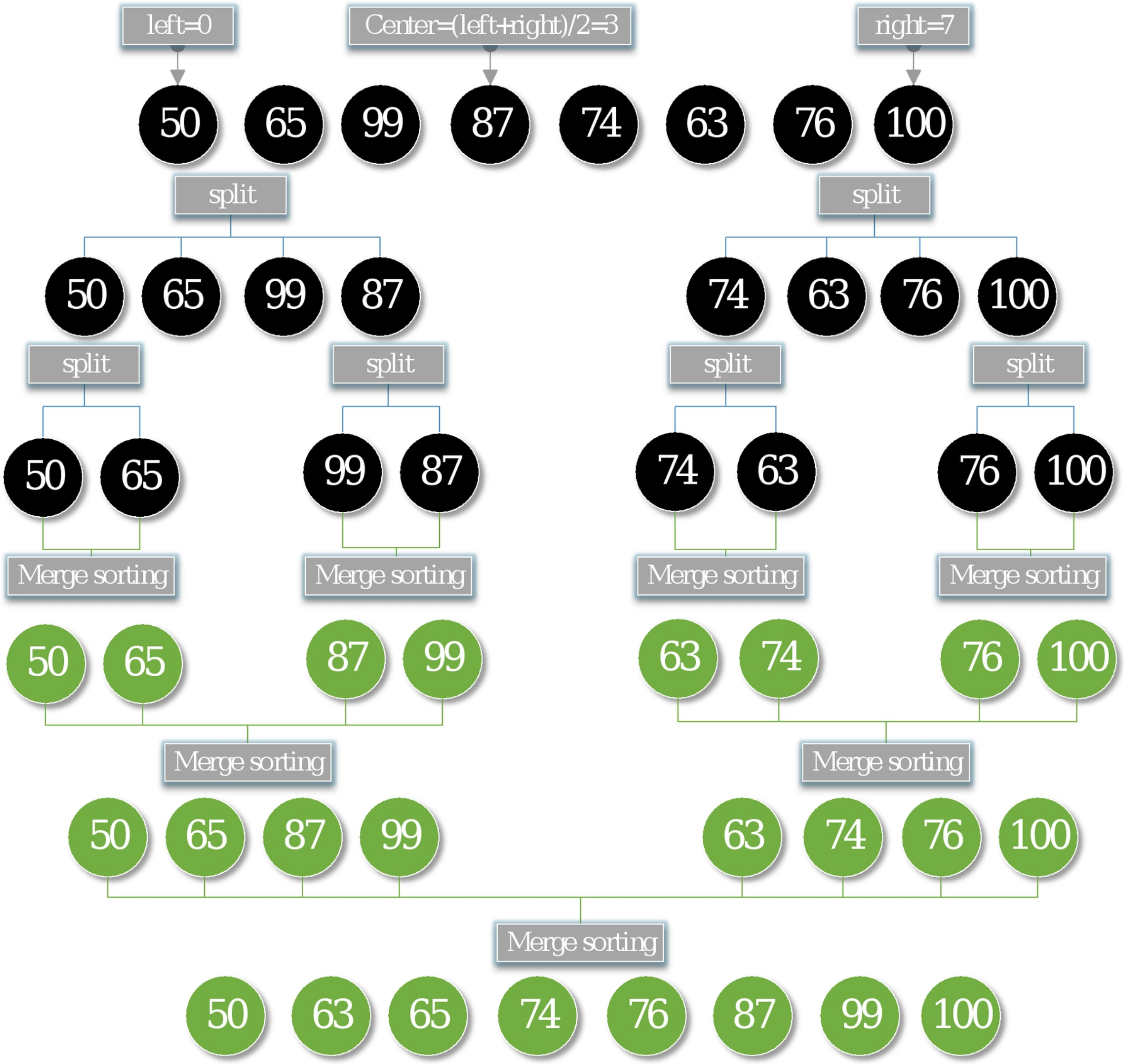

top
return =1;
pop
return =factorial(1)*2;
return =factorial(2)*3;
return =factorial(3)*4;
return =factorial(4)*5;
return =1;
top
return =1*2;
return =factorial(2)*3;
return =factorial(3)*4;
return =factorial(4)*5;
pop
return =1*2;
return =1*2*3;
pop
top
return =1*2*3;
return =factorial(3)*4;
return =factorial(4)*5;
top
return =1*2*3*4;
return =factorial(4)*5;
pop
return =1*2*3*4;
top
return =1*2*3*4*5;
pop
fac =factorial(5)=120

Two-way Merge Algorithm

**Two-way Merge Algorithm:**

The data of the first half and the second half are sorted, and the two ordered sub-list are merged into one ordered list, which continue to recursive to the end.

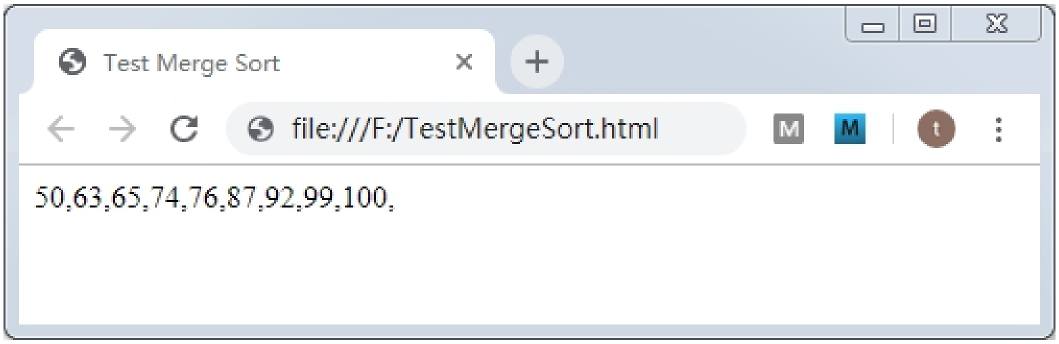
**1. The scores {50, 65, 99, 87, 74, 63, 76, 100} by merge sort**


left=0
Center=(left+right)/2=3
right=7
50
65
99
87
74
63
76
100
split
split
50
65
99
87
74
63
76
100
split
split
split
split
50
65
99
87
74
63
76
100
Merge sorting
Merge sorting
Merge sorting
Merge sorting
50
65
87
99
63
74
76
100
Merge sorting
Merge sorting
50
65
87
99
63
74
76
100
Merge sorting
50
63
65
74
76
87
99
100

**Create a TestMergeSort.html with Notepad**

```
<script type ="text/javascript" >
    class MergeSort{
        sort( array) {
            var temp = new Array( array.length);
            this.mergeSort( array, temp, 0 , array.length - 1 );
        }

        mergeSort( array, temp, left, right) {
            if ( left < right) {
                var center = parseInt(( left + right) / 2 );
                this.mergeSort( array, temp, left, center); // Left merge sort
                this.mergeSort( array, temp, center + 1 , right); // Right merge sort
                this.merge( array, temp, left, center + 1 , right); // Merge two ordered arrays
            }
        }

        /**
         Combine two ordered list into an ordered list
         temp : Temporary array
         left :    Start the subscript on the left
         right :  Start the subscript on the right
         rightEndIndex : End subscript on the right
         */
        merge( array, temp, left, right, rightEndIndex) {
            var leftEndIndex = right - 1 ; // End subscript on the left
            var tempIndex = left; // Starting from the left count
            var elementNumber = rightEndIndex - left + 1 ;

            while ( left <= leftEndIndex && right <= rightEndIndex) {
                if ( array[ left] <= array[ right])
                    temp[ tempIndex++] = array[ left++];
                else
                    temp[ tempIndex++] = array[ right++];
            }
            while ( left <= leftEndIndex) { // If there is element on the left
                temp[ tempIndex++] = array[ left++];
            }

            while ( right <= rightEndIndex) { // If there is element on the right
                temp[ tempIndex++] = array[ right++];
            }

            // Copy temp to array
            for ( var i = 0 ; i < elementNumber; i++) {
                array[ rightEndIndex] = temp[ rightEndIndex];
                rightEndIndex--;
            }
        }
    }

    //////////////////////testing////////////////////

    var scores = [ 50 , 65 , 99 , 87 , 74 , 63 , 76 , 100 , 92 ];
    var mergeSort = new MergeSort();
    mergeSort.sort( scores);
    for ( var i = 0 ; i < scores.length; i++) {
        document.write( scores[ i] + "," );
    }

</script>
```

**Result:**

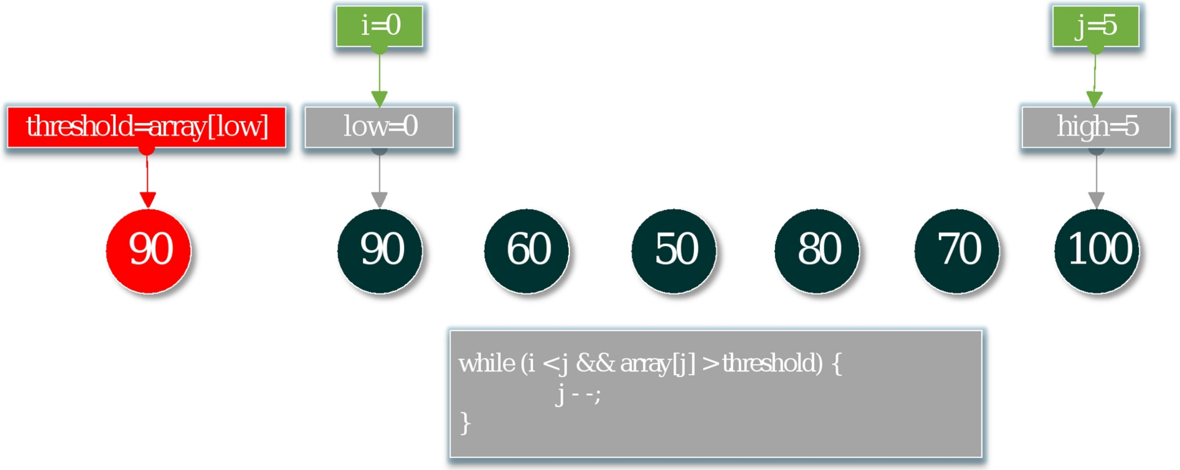

Test Merge Sort
file:///F:/TestMergeSort.html

```
50,63,65,74,76,87,92,99,100,
```

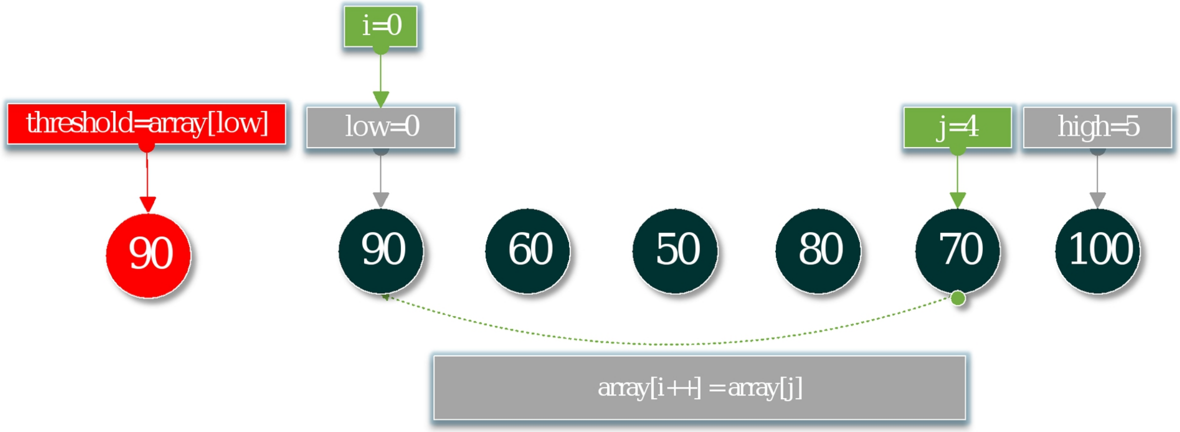
# Quick Sort Algorithm

**Quick Sort Algorithm:**
Divide the array into 2 subarrays, then call yourself recursively, sort the subarrays quickly, and continue to the end.

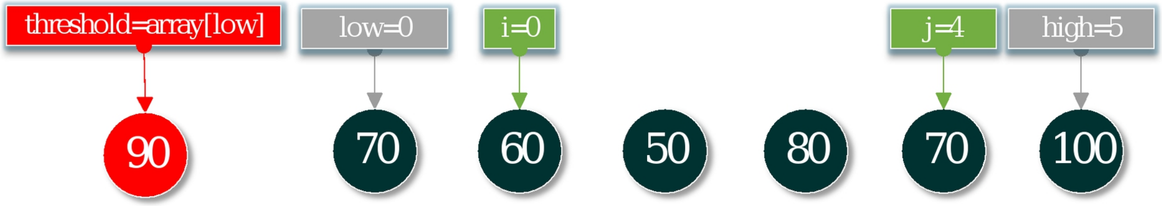
### 1. The scores {90, 60, 50, 80, 70, 100} by quick sort


i=0
j=5
threshold=array[low]
low=0
high=5
90
90
60
50
80
70
100

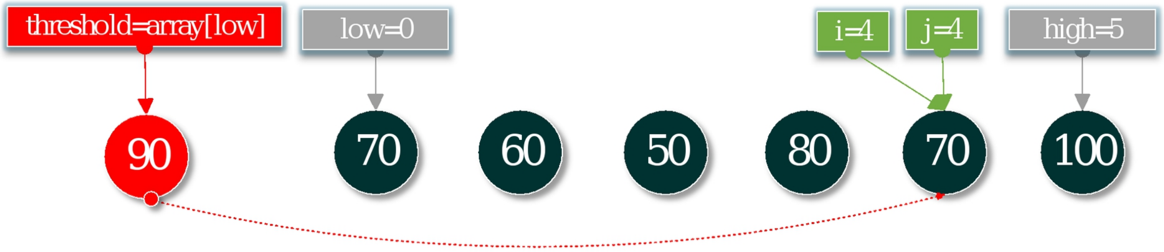
```
while (i <j && array[j] >threshold) {
            j - -;
}
```


i=0
threshold=array[low]
low=0
j=4
high=5
90
90
60
50
80
70
100

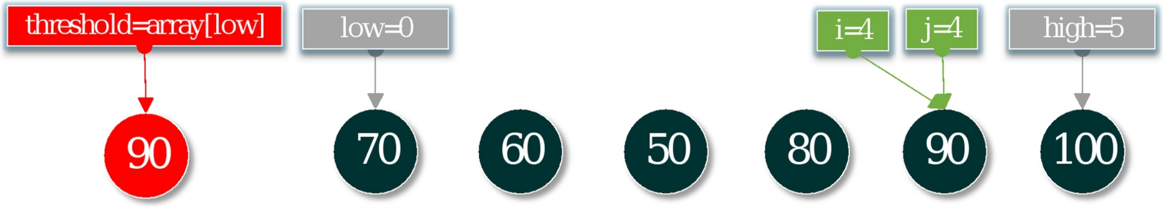
```
array[i++] =array[j]
```

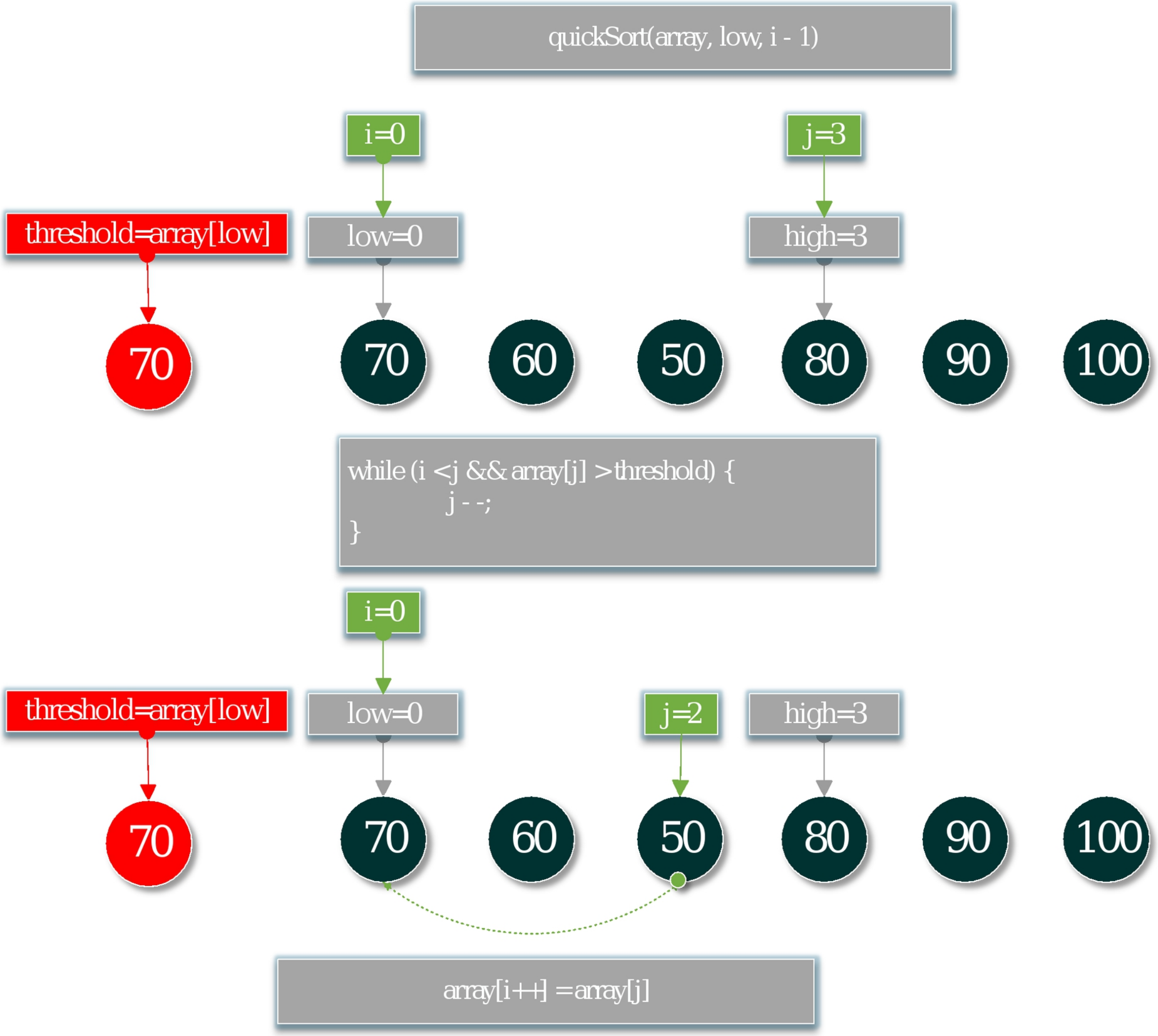

threshold=array[low]
low=0
i=0
j=4
high=5
90
70
60
50
80
70
100

```
while (i <j && array[i] <=threshold) {
        i++;
}
```


threshold=array[low]
low=0
i=4
j=4
high=5
90
70
60
50
80
70
100

```
array[i] = threshold
```

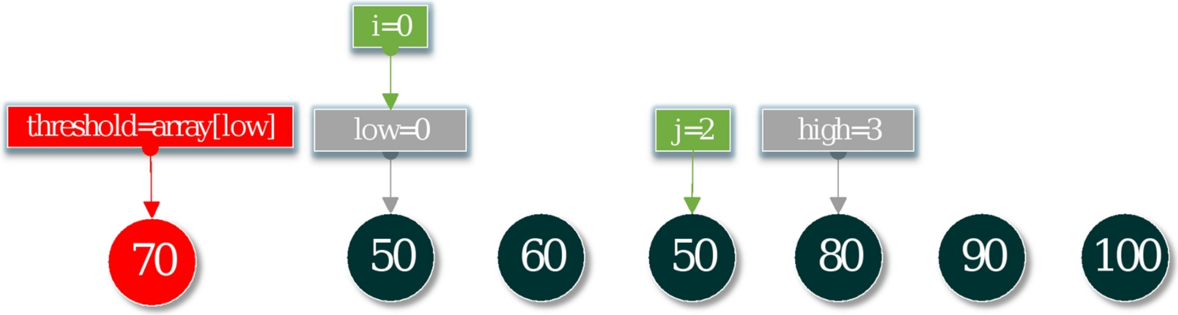

threshold=array[low]
low=0
i=4
j=4
high=5
90
70
60
50
80
90
100



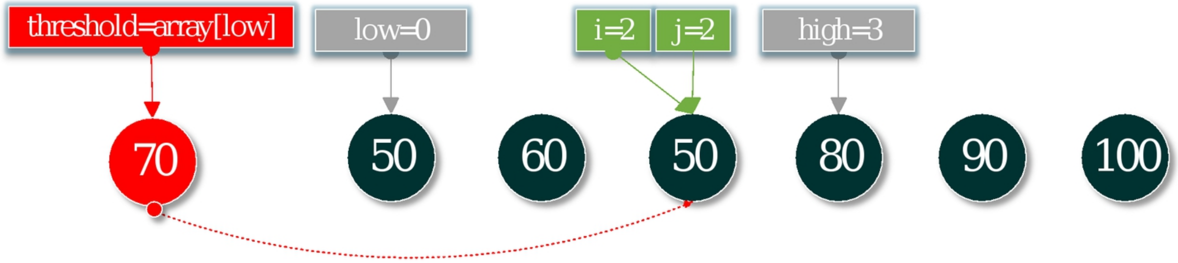
quickSort(array, low, i - 1)
i=0
j=3
threshold=array[low]
low=0
high=3
70
70
60
50
80
90
100
i=0
threshold=array[low]
low=0
j=2
high=3
70
70
60
50
80
90
100

```
while (i <j && array[j] >threshold) {
            j - -;
}
```

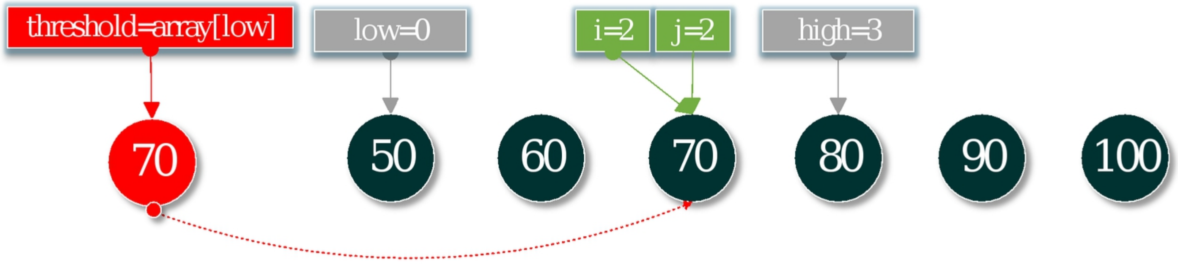
array[i++] =array[j]

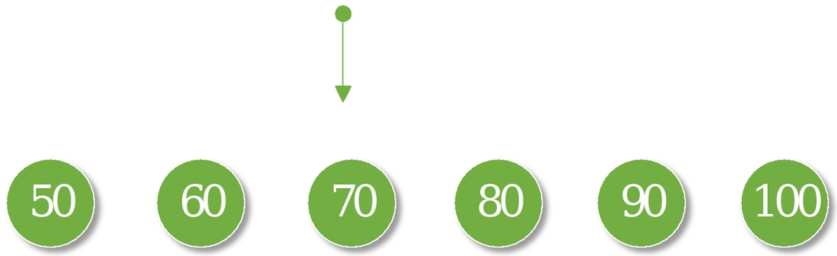

i=0
threshold=array[low]
low=0
j=2
high=3
70
50
60
50
80
90
100

```
while (i <j && array[i] <= threshold) {
        i++;
}
```


threshold=array[low]
low=0
i=2
j=2
high=3
70
50
60
50
80
90
100

```
array[i] = threshold
```

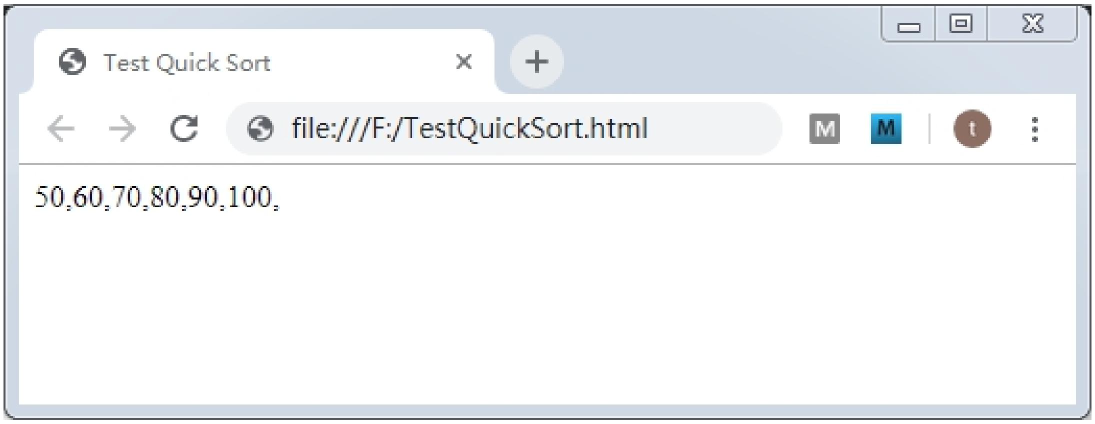

threshold=array[low]
low=0
i=2
j=2
high=3
70
50
60
70
80
90
100


50
60
70
80
90
100

**Create a TestQuickSort.html with Notepad**

```
<script type ="text/javascript" >
    class QuickSort{
        sort( array) {
            if ( array.length > 0 ) {
                this.quickSort( array, 0 , array.length - 1 );
            }
        }

        quickSort( array, low, high) {
            if ( low > high) {
                return ;
            }
            var i = low;
            var j = high;
            var threshold = array[ low];
            // Alternately scanned from both ends of the list
            while ( i < j) {
                // Find the first position less than threshold from right to left
                while ( i < j && array[ j] > threshold) {
                    j--;
                }
                //Replace the low with a smaller number than the threshold
                if ( i < j)
                    array[ i++] = array[ j];

                // Find the first position greater than threshold from left to right
                while ( i < j && array[ i] <= threshold) {
                    i++;
                }

                //Replace the high with a number larger than the threshold
                if ( i < j)
                    array[ j--] = array[ i];
            }
            array[ i] = threshold;
            // left quickSort
            this.quickSort( array, low, i - 1 );
            // right quickSort
            this.quickSort( array, i + 1 , high);
        }
    }

    ////////////////////testing/////////////////

    var scores = [ 90 , 60 , 50 , 80 , 70 , 100 ];
    var quickSort = new QuickSort();
    quickSort.sort( scores);
    for ( var i = 0 ; i < scores.length; i++) {
        document.write( scores[ i] + "," );
    }
</script>
```

**Result:**

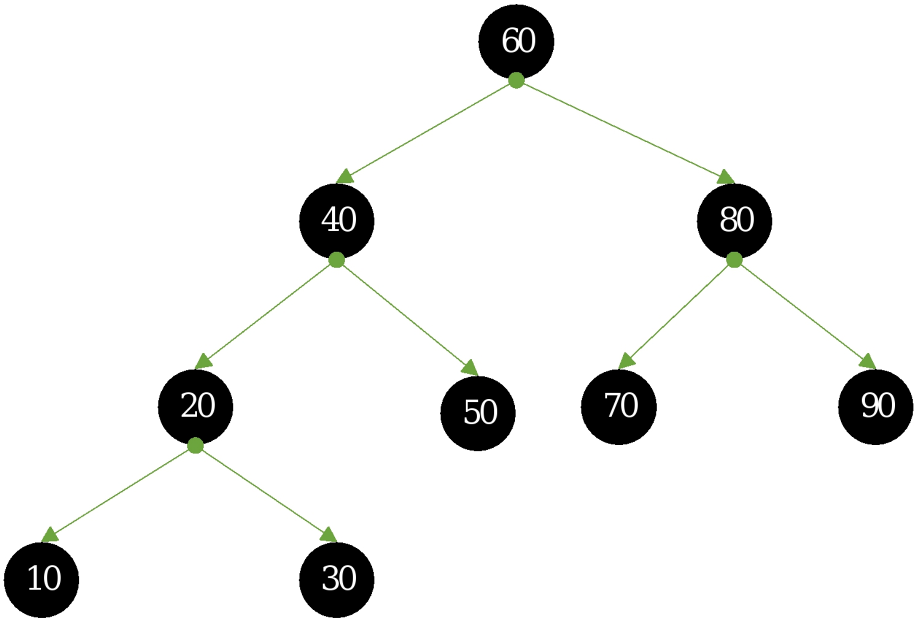

Test Quick Sort
file:///F:/TestQuickSort.html

```
50,60,70,80,90,100,
```

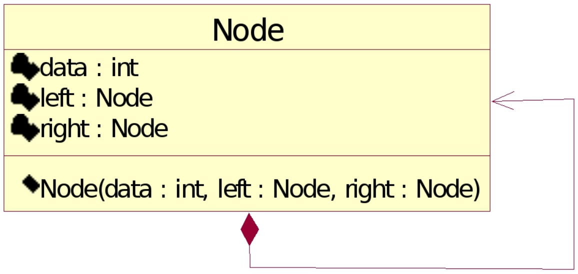
# Binary Search Tree

**Binary Search Tree:**

1. If the left subtree of any node is not empty, the value of all nodes on the left subtree is less than the value of its root node;
2. If the right subtree of any node is not empty, the value of all nodes on the right subtree is greater than the value of its root node;
3. The left subtree and the right subtree of any node are also binary search trees.


60
40
80
20
50
70
90
10
30

### Node UML Diagram

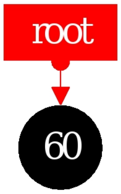

Node
data : int
left : Node
right : Node
Node(data : int, left : Node, right : Node)

```
class Node{
    constructor( data, left, right){
        this.data = data;
        this.left = left;
        this.right = right;
    }

    getData(){
        return this.data;
    }
}
```

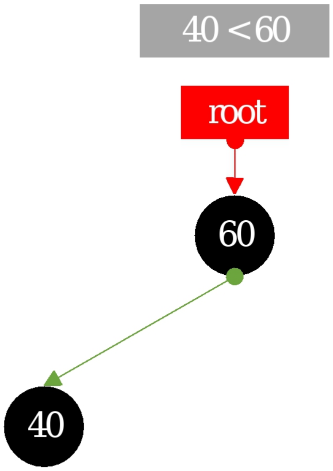
## 1.Construct a binary search tree, insert node

The inserted nodes are compared from the root node, and the smaller than the root node is compared with the left subtree of the root node, otherwise, compared with the right subtree until the left subtree is empty or the right subtree is empty, then is inserted.

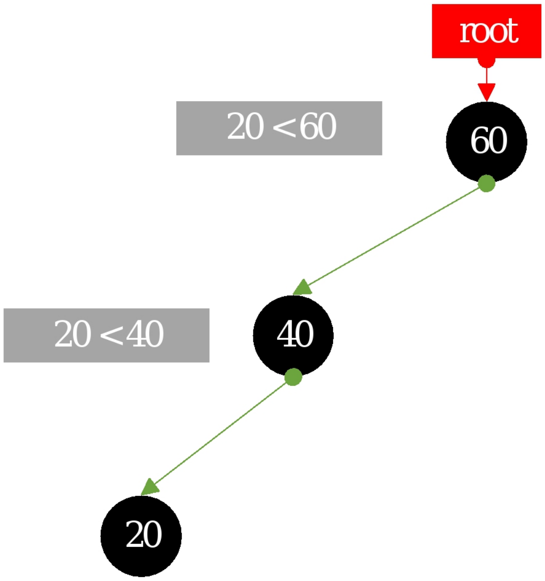
**Insert 60**



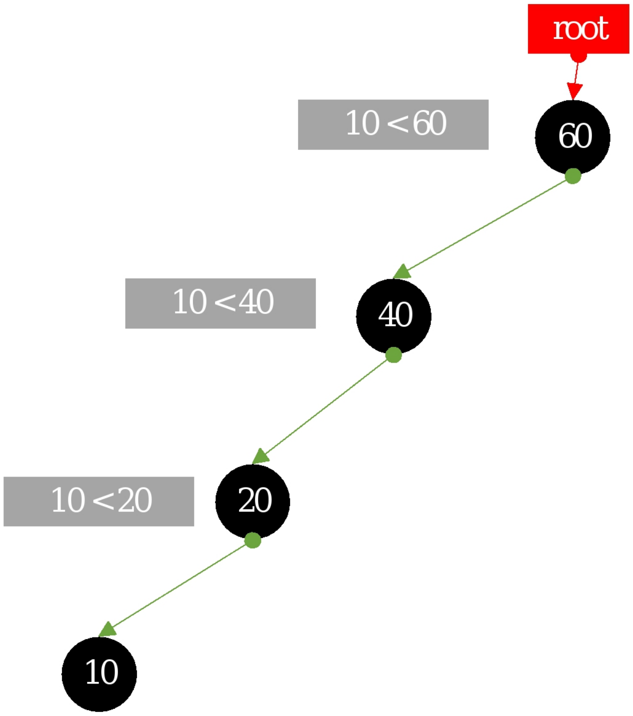
**Insert 40**


40 < 60
root
60
40

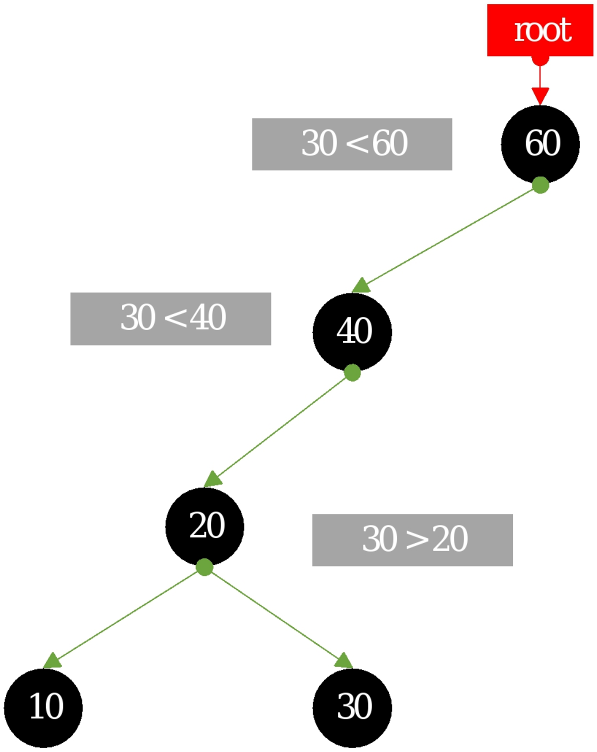
**Insert** **20**


root
20 < 60
60
20 < 40
40
20

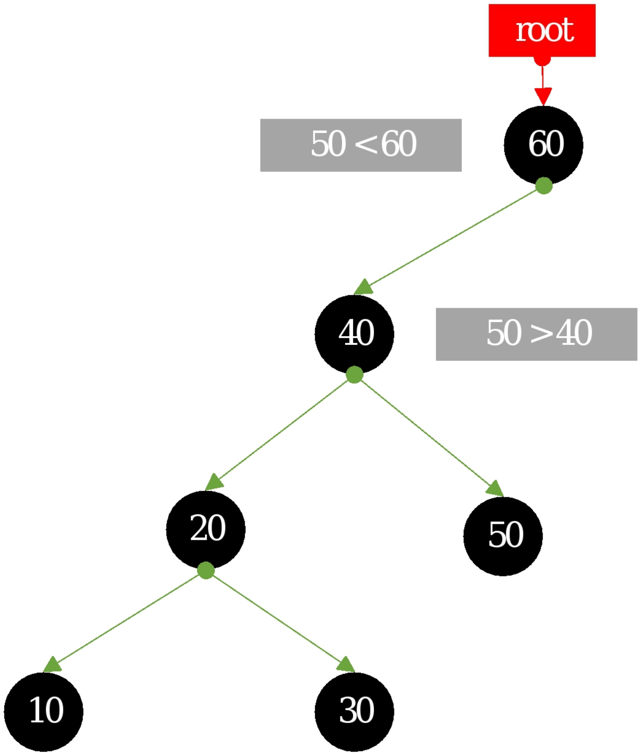
**Insert** **10**


root
10 < 60
60
10 < 40
40
10 < 20
20
10

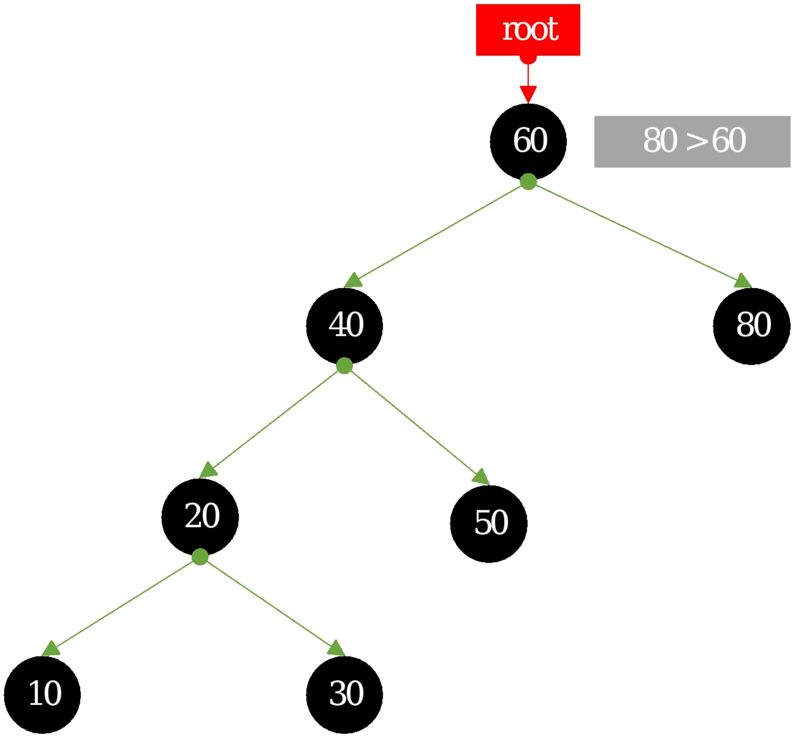
**Insert 30**


root
30 < 60
60
30 < 40
40
20
30 > 20
10
30

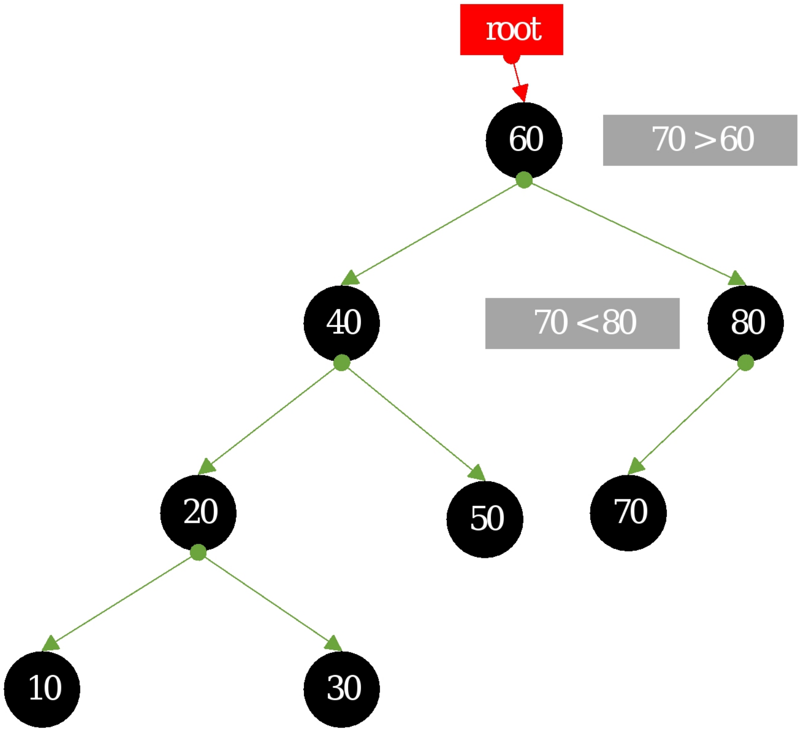
**Insert 50**


root
50 <60
60
40
50 >40
20
50
10
30

**Insert 80**

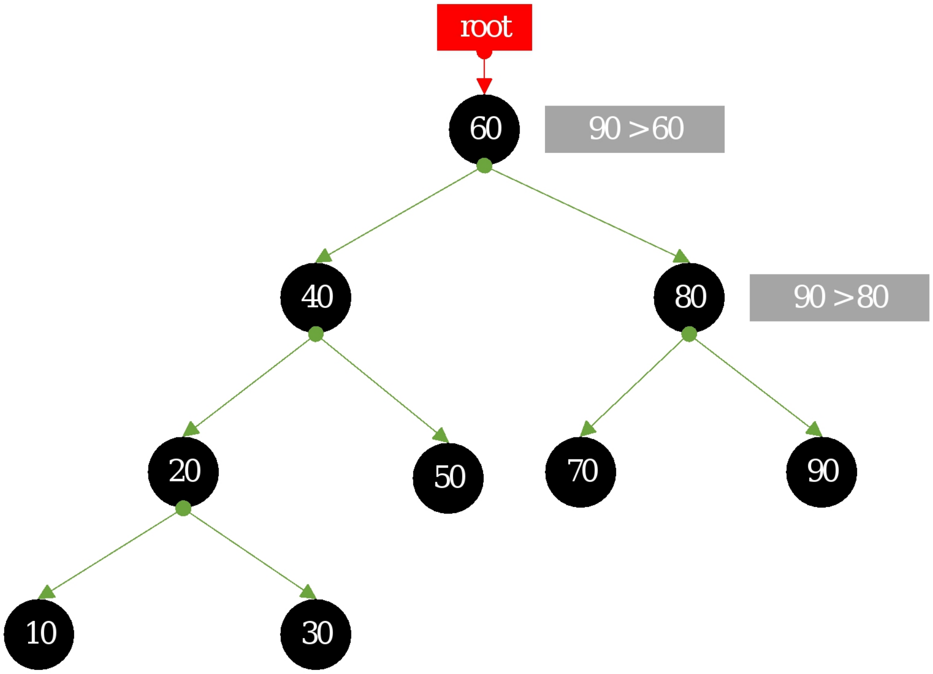

root
60
80 >60
40
80
20
50
10
30

**Insert 70**



root
70 > 60
70 < 80

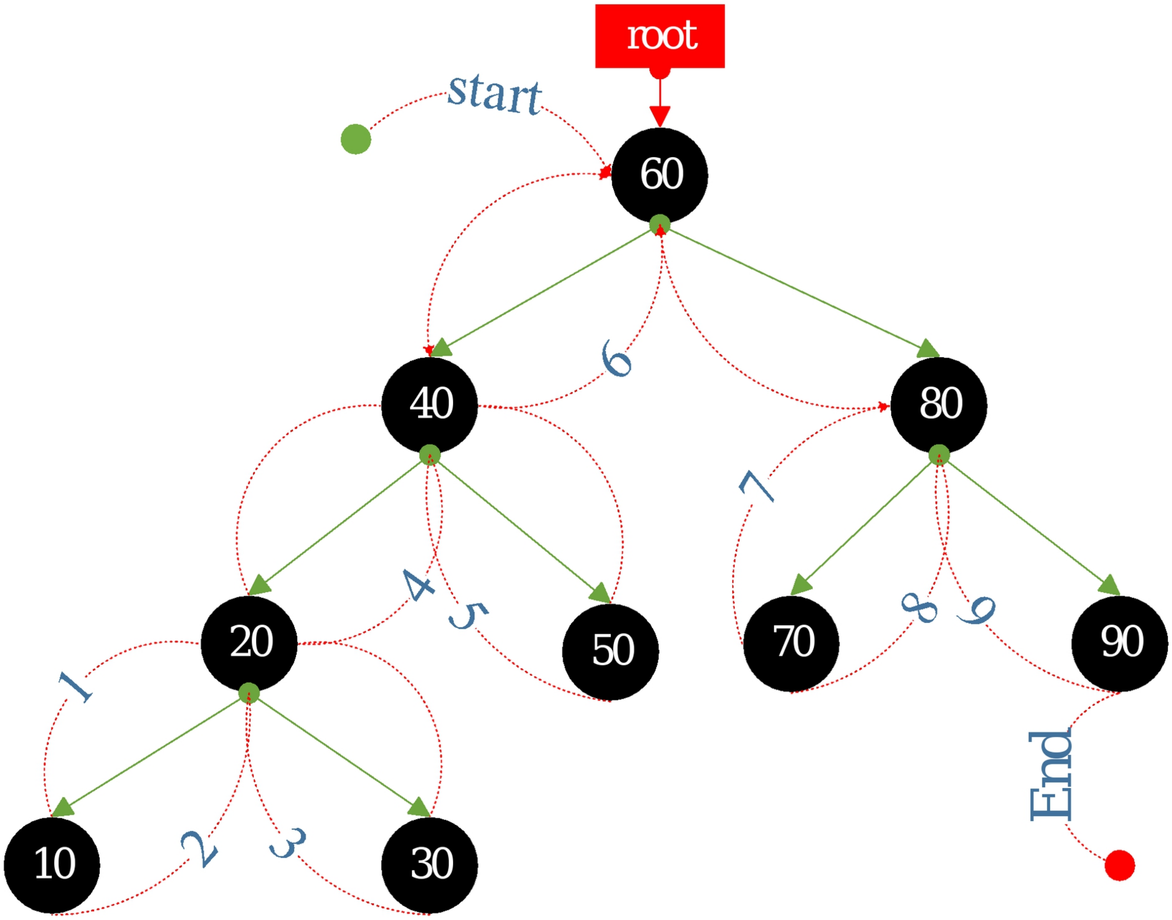
**Insert 90**



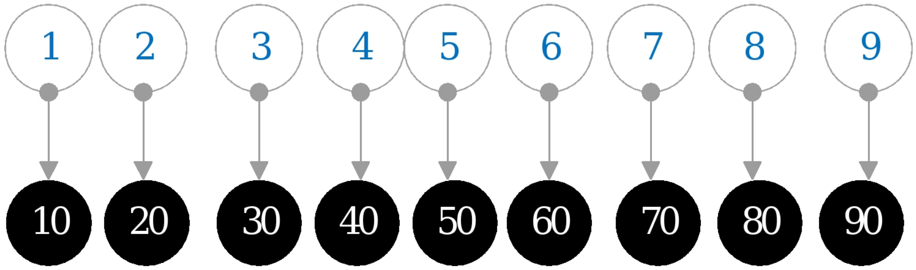
root
90 > 60
90 > 80

## 2. binary search tree In-order traversal

**In-order traversal** : left subtree -> root node -> right subtree


root
start
60
40
80
6
7
20
4
5
50
70
8
9
90
1
End
10
2
3
30

**Result:**

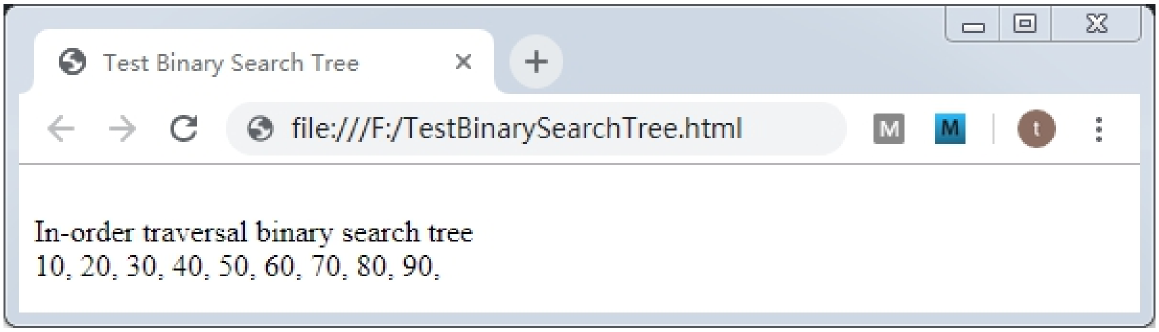

1
2
3
4
5
6
7
8
9
10
20
30
40
50
60
70
80
90

**Create a TestBinarySearchTreeInOrder.html with Notepad**

```
<script type ="text/javascript" >
    class BinaryTree {
        getRoot() {
            return this.root;
        }

        // In-order traversal binary search tree
        inOrder( root) {
            if ( root == null ) {
                return ;
            }
            this.inOrder( root.left); // Traversing the left subtree
            document.write( root.getData() + ", " );
            this.inOrder( root.right); // Traversing the right subtree
        }

        insert( node, newData) {
            if ( this.root == null ) {
                this.root = new Node( newData, null , null );
                return ;
            }

            var compareValue = newData - node.getData();

            //Recursive left subtree, continue to find the insertion position
            if ( compareValue < 0 ) {
                if ( node.left == null ) {
                    node.left = new Node( newData, null , null );
                } else {
                    this.insert( node.left, newData);
                }
            } else if ( compareValue > 0 ) { ////Recursive right subtree, continue to find the insertion position
                if ( node.right == null ) {
                    node.right = new Node( newData, null , null );
                } else {
```

```
                this.insert( node.right, newData);
            }
        }
    }
}


class Node{
    constructor( data, left, right){
        this.data = data;
        this.left = left;
        this.right = right;
    }

    getData(){
        return this.data;
    }
}

////////////////////testing/////////////////

var binaryTree= new BinaryTree();
//Constructing a binary search tree
binaryTree.insert( binaryTree.getRoot(), 60 );
binaryTree.insert( binaryTree.getRoot(), 40 );
binaryTree.insert( binaryTree.getRoot(), 20 );
binaryTree.insert( binaryTree.getRoot(), 10 );
binaryTree.insert( binaryTree.getRoot(), 30 );
binaryTree.insert( binaryTree.getRoot(), 50 );
binaryTree.insert( binaryTree.getRoot(), 80 );
binaryTree.insert( binaryTree.getRoot(), 70 );
binaryTree.insert( binaryTree.getRoot(), 90 );
document.write( "<br> In-order traversal binary search tree <br>" );
binaryTree.inOrder( binaryTree.getRoot());
</script>
```

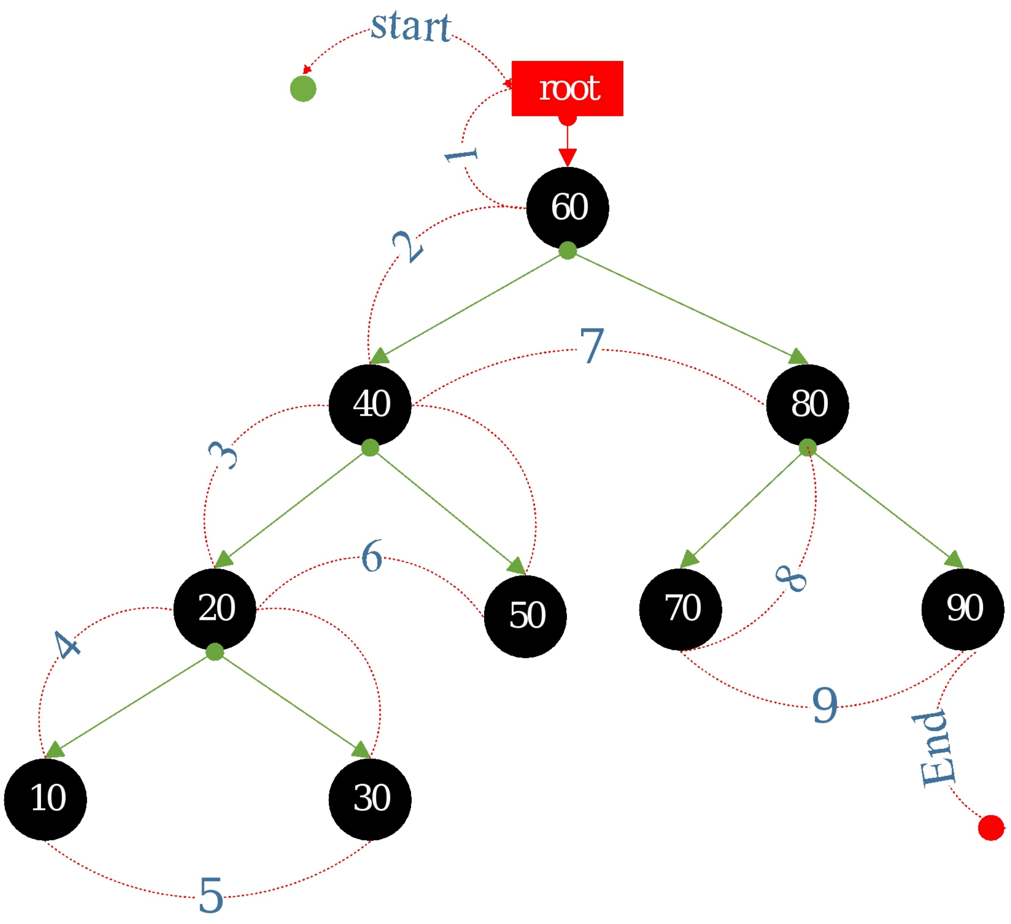
**Result:**


Test Binary Search Tree
file:///F:/TestBinarySearchTree.html

```
In-order traversal binary search tree
10, 20, 30, 40, 50, 60, 70, 80, 90,
```

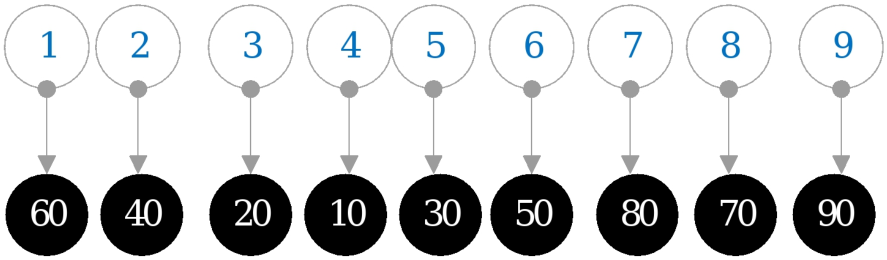
## 3. binary search tree **Pre-order traversal**

**Pre-order traversal** : root node -> left subtree -> right subtree



start
root
1
2
3
4
5
6
7
8
9
End
60
40
80
20
50
70
90
10
30

**Result:**

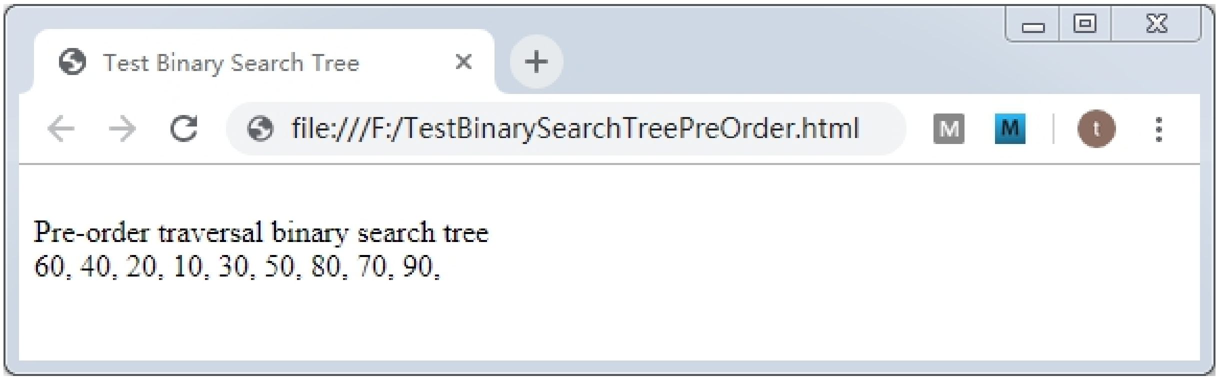


1
2
3
4
5
6
7
8
9
60
40
20
10
30
50
80
70
90

**Create a TestBinarySearchTreePreOrder.html with Notepad**

```
<script type ="text/javascript" >
    class BinaryTree {
        getRoot() {
            return this.root;
        }

        //Preorder traversal binary search tree
        preOrder( root) {
            if ( root == null ) {
                return ;
            }
            document.write( root.getData() + ", " );
            this.preOrder( root.left); // Recursive Traversing the left subtree
            this.preOrder( root.right); // Recursive Traversing the right subtree
        }

        insert( node, newData) {
            if ( this.root == null ) {
                this.root = new Node( newData, null , null );
                return ;
            }

            var compareValue = newData - node.getData();
            //Recursive left subtree, continue to find the insertion position
            if ( compareValue < 0 ) {
                if ( node.left == null ) {
                    node.left = new Node( newData, null , null );
                } else {
                    this.insert( node.left, newData);
                }
            } else if ( compareValue > 0 ) { ////Recursive right subtree, continue
to find the insertion position
                if ( node.right == null ) {
                    node.right = new Node( newData, null , null );
                } else {
                    this.insert( node.right, newData);
                }
            }
        }
    }


    class Node{
        constructor( data, left, right){
            this.data = data;
            this.left = left;
            this.right = right;
        }

        getData(){
            return this.data;
        }
    }

    ////////////////////testing//////////////////

    var binaryTree= new BinaryTree();
    //Constructing a binary search tree
    binaryTree.insert( binaryTree.getRoot(), 60 );
    binaryTree.insert( binaryTree.getRoot(), 40 );
    binaryTree.insert( binaryTree.getRoot(), 20 );
    binaryTree.insert( binaryTree.getRoot(), 10 );
    binaryTree.insert( binaryTree.getRoot(), 30 );
    binaryTree.insert( binaryTree.getRoot(), 50 );
    binaryTree.insert( binaryTree.getRoot(), 80 );
    binaryTree.insert( binaryTree.getRoot(), 70 );
    binaryTree.insert( binaryTree.getRoot(), 90 );

    document.write( "<br> Pre-order traversal binary search tree <br>" );
    binaryTree.preOrder( binaryTree.getRoot());
</script>
```

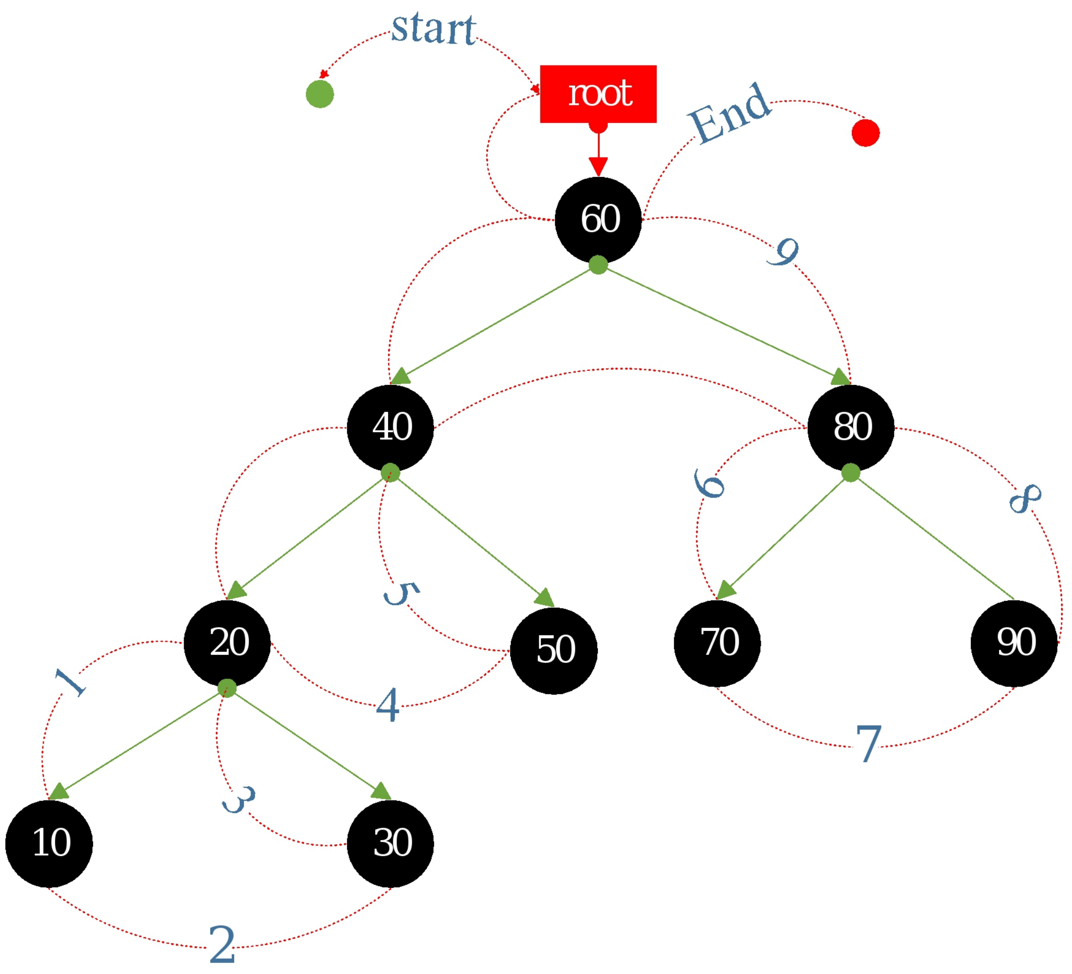
**Result:**


Test Binary Search Tree
file:///F:/TestBinarySearchTreePreOrder.html

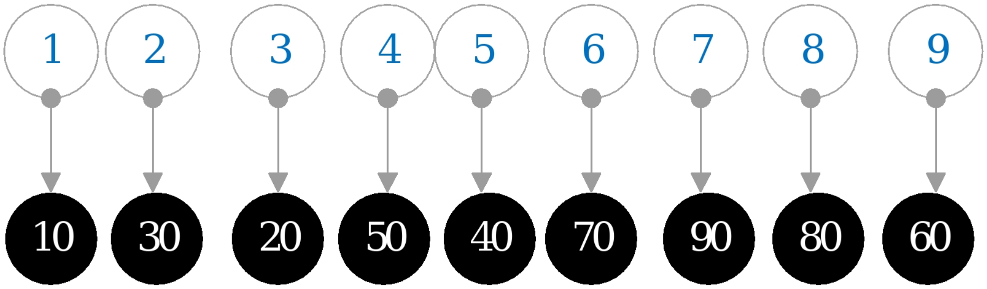
```
Pre-order traversal binary search tree
60, 40, 20, 10, 30, 50, 80, 70, 90,
```

## 4. binary search tree Post-order traversal

**Post-order traversal** : right subtree -> root node -> left subtree


start
root
End
60
40
80
20
50
70
90
10
30
1
2
3
4
5
6
7
8
9

**Result:**

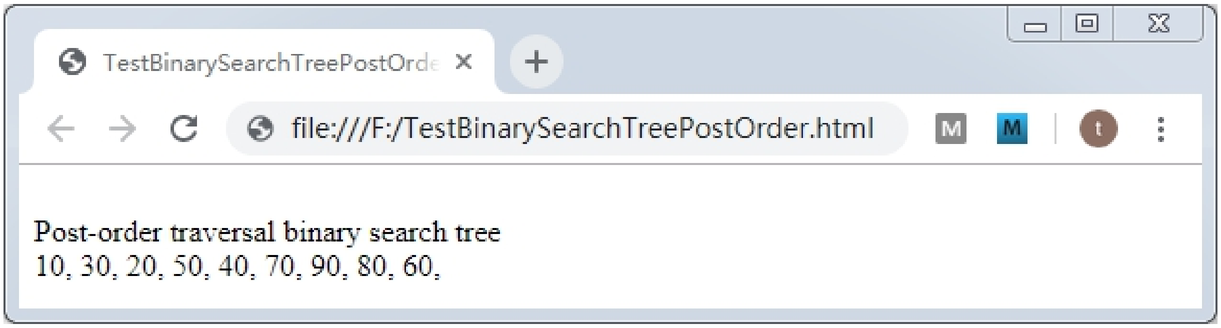

1
2
3
4
5
6
7
8
9
10
30
20
50
40
70
90
80
60

**Create a TestBinarySearchTreePostOrder.html with Notepad**

```
<script type ="text/javascript" >
   class BinaryTree {
      getRoot() {
         return this.root;
      }

      //Post-order traversal binary search tree
      postOrder( root) {
         if ( root == null ) {
            return ;
         }
         this.postOrder( root.left); // Recursive Traversing the left subtree
         this.postOrder( root.right); // Recursive Traversing the right subtree
         document.write( root.getData() + ", " );
      }

      insert( node, newData) {
         if ( this.root == null ) {
            this.root = new Node( newData, null , null );
            return ;
         }

         var compareValue = newData - node.getData();
         //Recursive left subtree, continue to find the insertion position
         if ( compareValue < 0 ) {
            if ( node.left == null ) {
               node.left = new Node( newData, null , null );
            } else {
               this.insert( node.left, newData);
            }
         } else if ( compareValue > 0 ) { ////Recursive right subtree, continue
to find the insertion position
            if ( node.right == null ) {
               node.right = new Node( newData, null , null );
            } else {
                this.insert( node.right, newData);
            }
        }
    }
}


class Node{
    constructor( data, left, right){
        this.data = data;
        this.left = left;
        this.right = right;
    }

    getData(){
        return this.data;
    }
}

//////////////////////testing////////////////////

var binaryTree= new BinaryTree();

//Constructing a binary search tree
binaryTree.insert( binaryTree.getRoot(), 60 );
binaryTree.insert( binaryTree.getRoot(), 40 );
binaryTree.insert( binaryTree.getRoot(), 20 );
binaryTree.insert( binaryTree.getRoot(), 10 );
binaryTree.insert( binaryTree.getRoot(), 30 );
binaryTree.insert( binaryTree.getRoot(), 50 );
binaryTree.insert( binaryTree.getRoot(), 80 );
binaryTree.insert( binaryTree.getRoot(), 70 );
binaryTree.insert( binaryTree.getRoot(), 90 );

document.write( "<br> Post-order traversal binary search tree <br>" );
binaryTree.postOrder( binaryTree.getRoot());
```

```
</script>
```

**Result:**

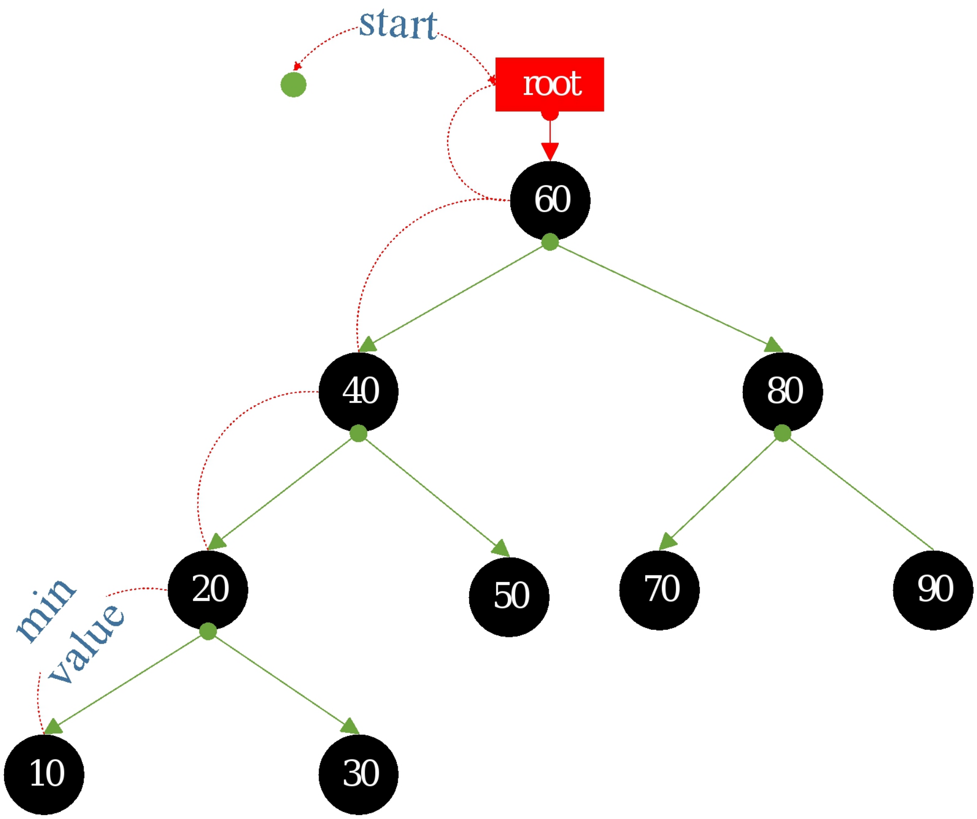

TestBinarySearchTreePostOrde
file:///F:/TestBinarySearchTreePostOrder.html

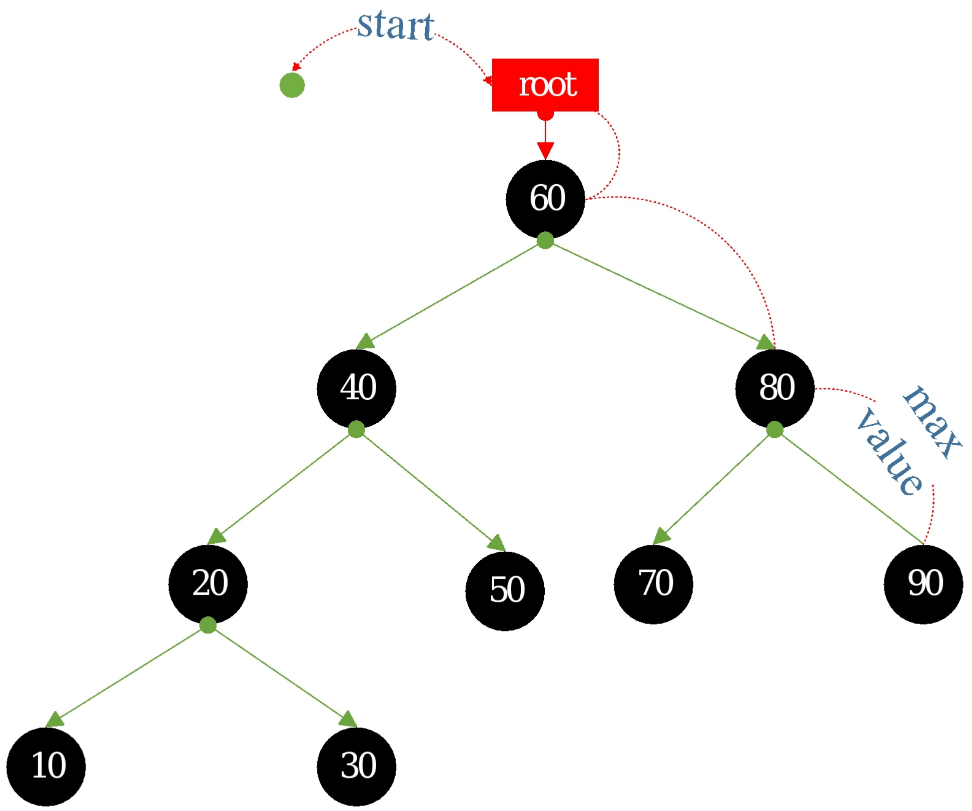
```
Post-order traversal binary search tree
10, 30, 20, 50, 40, 70, 90, 80, 60,
```

## 5. binary search tree Maximum and minimum

**Minimum value:** The small value is on the left child node, as long as the recursion traverses the left child until be empty, the current node is the minimum node.

**Maximum value:** The large value is on the right child node, as long as the recursive traversal is the right child until be empty, the current node is the largest node.


start
root
60
40
80
20
50
70
90
min value
10
30

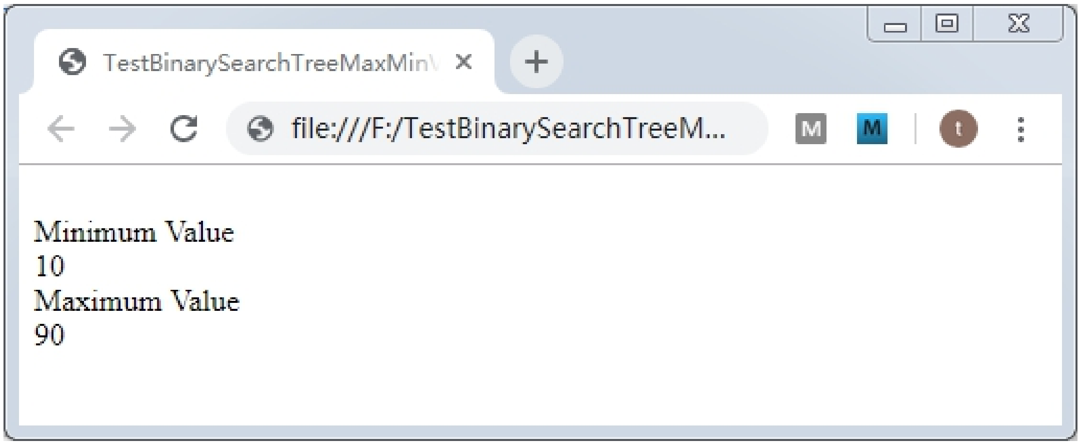

start
root
max
value

**Create a TestBinarySearchTreeMaxMinValue.html with Notepad**

```
<script type ="text/javascript" >
    class BinaryTree {
        getRoot() {
            return this.root;
        }

        //Minimum value
        searchMinValue( node) {
            if ( node == null || node.getData() == 0 )
                return null ;
            if ( node.left == null ) {
                return node;
            }
            //Recursively find the minimum from the left subtree
            return this.searchMinValue( node.left);
        }

        //Maximum value
        searchMaxValue( node) {
            if ( node == null || node.getData() == 0 )
                return null ;
            if ( node.right == null ) {
                return node;
            }
            //Recursively find the minimum from the right subtree
            return this.searchMaxValue( node.right);
        }
    insert( node, newData) {
        if ( this.root == null ) {
            this.root = new Node( newData, null , null );
            return ;
        }

        var compareValue = newData - node.getData();
        //Recursive left subtree, continue to find the insertion position
        if ( compareValue < 0 ) {
            if ( node.left == null ) {
                node.left = new Node( newData, null , null );
            } else {
                this.insert( node.left, newData);
            }
        } else if ( compareValue > 0 ) { ////Recursive right subtree, continue to find the insertion position
            if ( node.right == null ) {
                node.right = new Node( newData, null , null );
            } else {
                this.insert( node.right, newData);
            }
        }
    }
}

class Node{
    constructor( data, left, right){
        this.data = data;
        this.left = left;
        this.right = right;
    }

    getData(){
        return this.data;
    }
}









//////////////////////testing////////////////////

var binaryTree= new BinaryTree();

//Constructing a binary search tree
binaryTree.insert( binaryTree.getRoot(), 60 );
binaryTree.insert( binaryTree.getRoot(), 40 );
binaryTree.insert( binaryTree.getRoot(), 20 );
binaryTree.insert( binaryTree.getRoot(), 10 );
binaryTree.insert( binaryTree.getRoot(), 30 );
binaryTree.insert( binaryTree.getRoot(), 50 );
binaryTree.insert( binaryTree.getRoot(), 80 );
binaryTree.insert( binaryTree.getRoot(), 70 );
binaryTree.insert( binaryTree.getRoot(), 90 );

document.write( "<br> Minimum Value<br> " );
var minNode = binaryTree.searchMinValue( binaryTree.getRoot());
document.write( minNode.getData());

document.write( "<br> Maximum Value<br> " );
var maxNode = binaryTree.searchMaxValue( binaryTree.getRoot());
document.write( maxNode.getData());
```

```
</script>
```

**Result:**

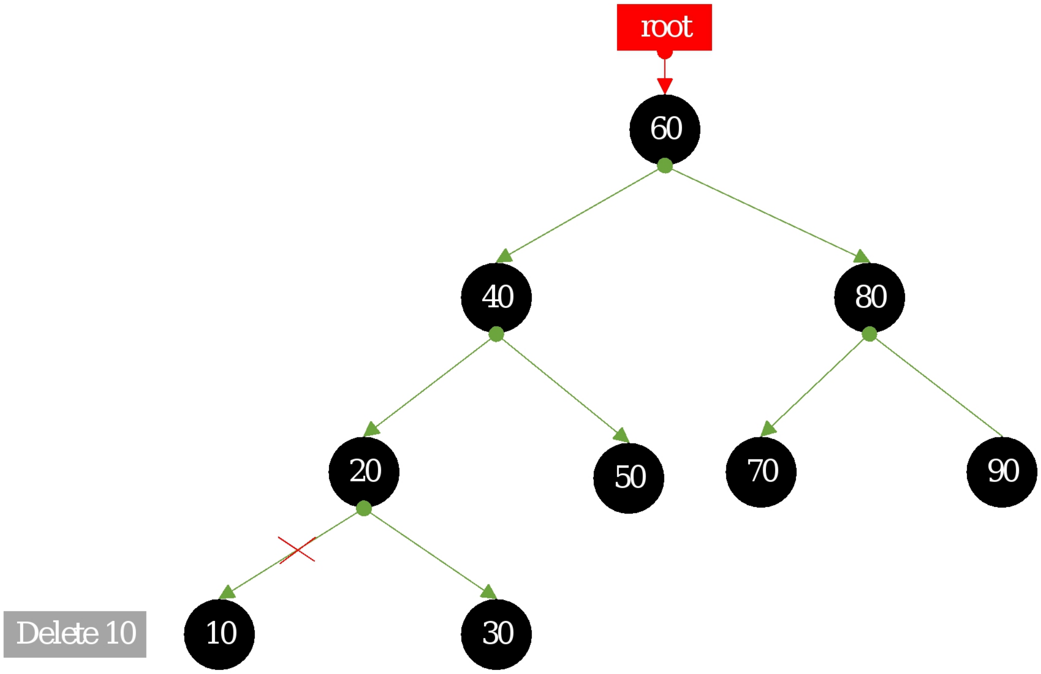

TestBinarySearchTreeMaxMin\
file:///F:/TestBinarySearchTreeM...

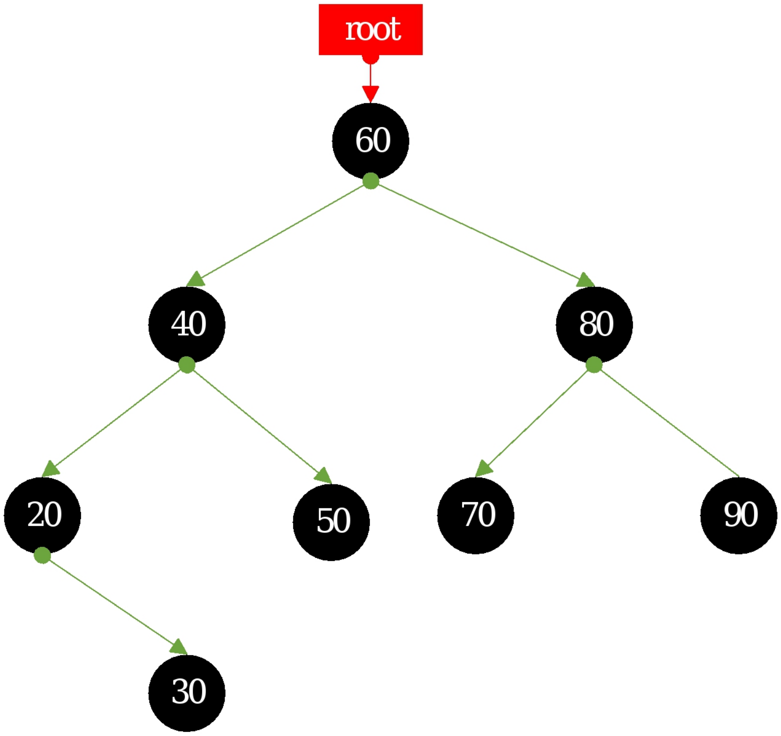
```
Minimum Value
10
Maximum Value
90
```

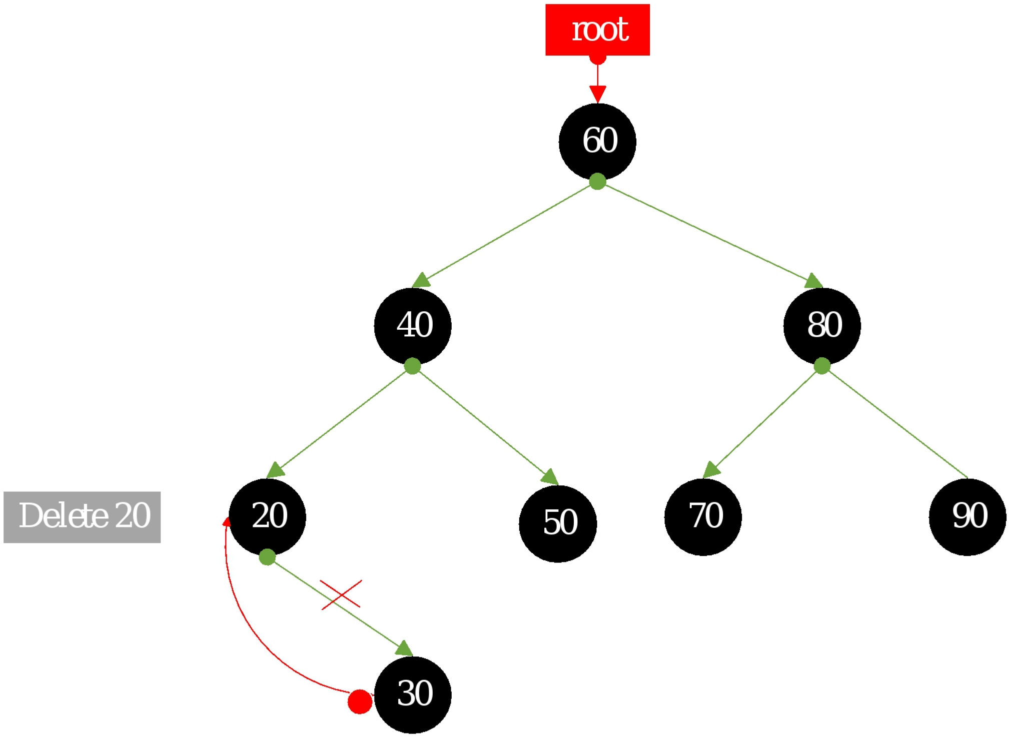
## 6. binary search tree **Delete Node**

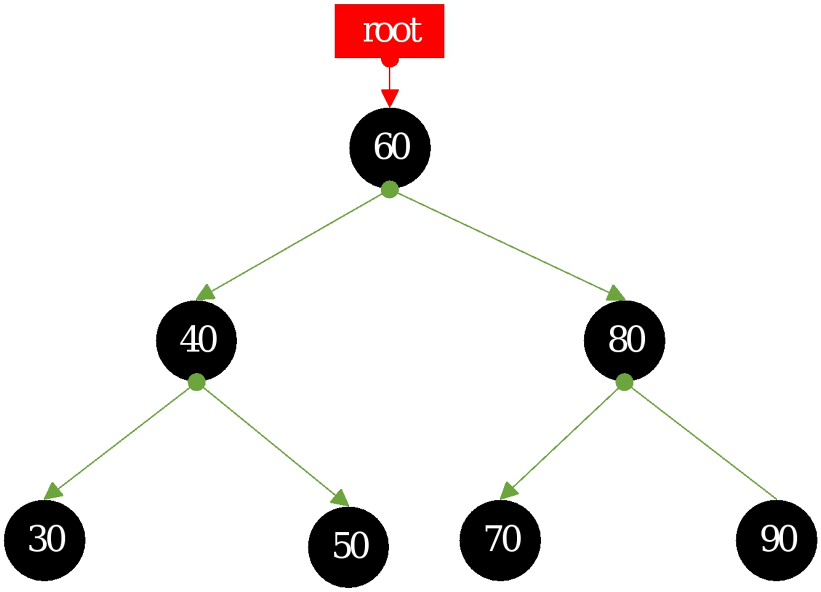
Binary search tree delete node 3 cases

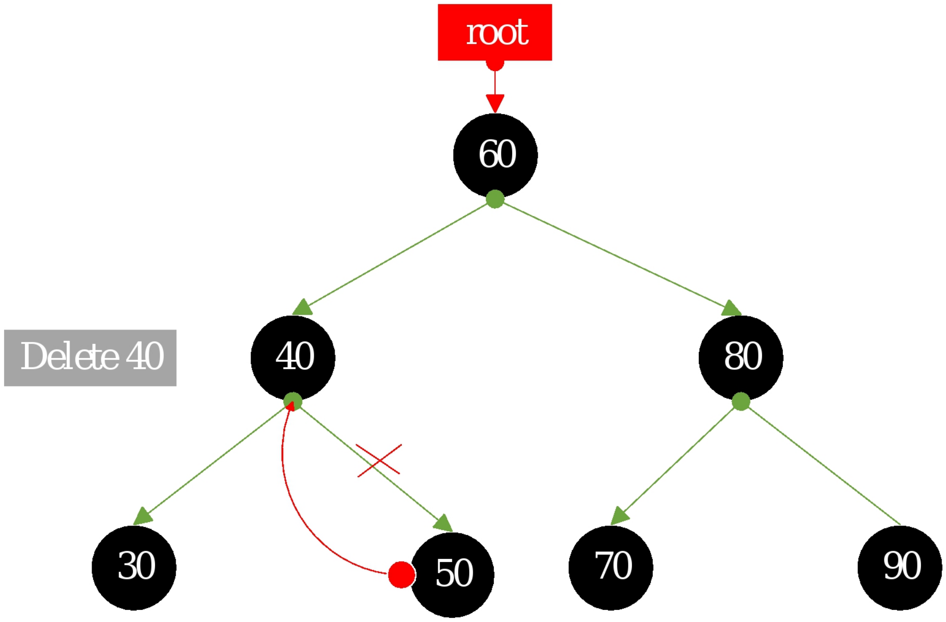
1. If there is no child node, delete it directly
2. If there is only one child node, the child node replaces the current node, and then deletes the current node.
3. If there are two child nodes, replace the current node with the smallest node from the right subtree, because the smallest node on the right is also larger than the value on the left.

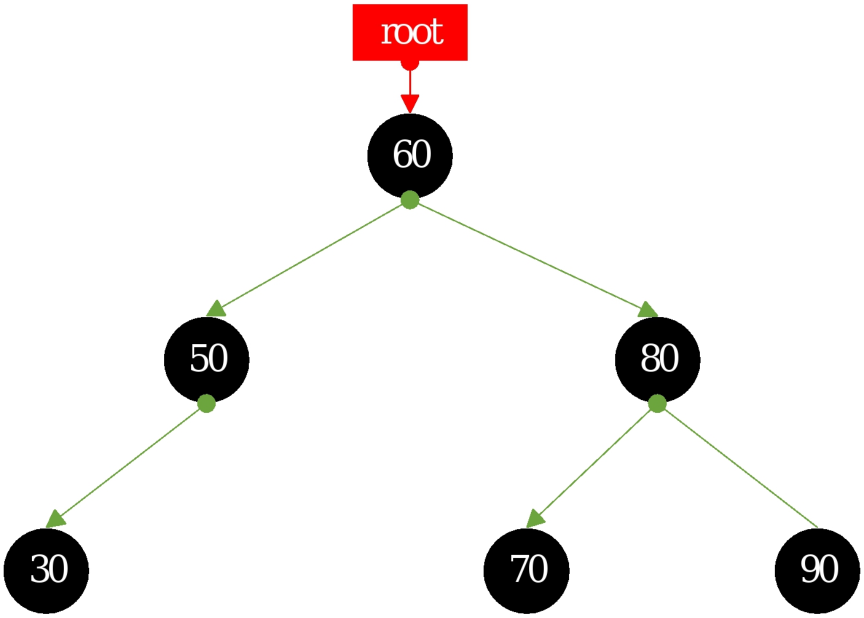



root


root
Delete 20


root


root
Delete 40


root

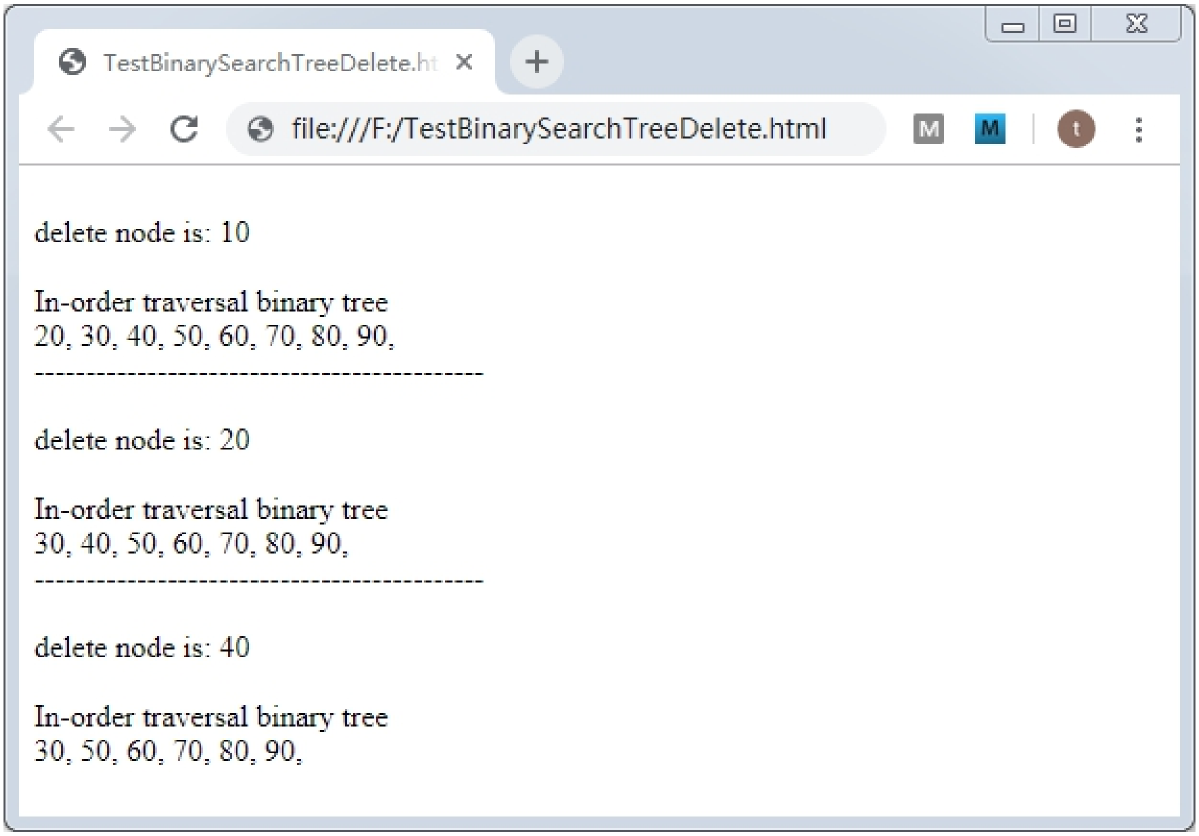
**Create a TestBinarySearchTreeDelete.html with Notepad**

```
<script type ="text/javascript" >
    class BinaryTree {
        getRoot() {
            return this.root;
        }

         // In-order traversal binary search tree
        inOrder( root) {
            if ( root == null ) {
                return ;
            }
            this.inOrder( root.left); // Traversing the left subtree
            document.write( root.getData() + ", " );
            this.inOrder( root.right); // Traversing the right subtree
        }

        remove( node, newData) {
            if ( node == null )
                return node;
            var compareValue = newData - node.getData();
            if ( compareValue > 0 ) {
                node.right = this.remove( node.right, newData);
            } else if ( compareValue < 0 ) {
                node.left = this.remove( node.left, newData);
            } else if ( node.left != null && node.right != null ) {
                //Find the minimum node of the right subtree to replace the current node
                node.setData( this.searchMinValue( node.right). getData());
                node.right = this.remove( node.right, node.getData());
            } else {
                node = ( node.left != null ) ? node.left : node.right;
            }
            return node;
        }
```

```
//Search minimum
searchMinValue( node) {
    if ( node == null || node.getData() == 0 )
        return null ;
    if ( node.left == null ) {
        return node;
    }
    //Recursively find the minimum from the left subtree
    return this.searchMinValue( node.left);
}

insert( node, newData) {
    if ( this.root == null ) {
        this.root = new Node( newData, null , null );
        return ;
    }

    var compareValue = newData - node.getData();
    //Recursive left subtree, continue to find the insertion position
    if ( compareValue < 0 ) {
        if ( node.left == null ) {
            node.left = new Node( newData, null , null );
        } else {
            this.insert( node.left, newData);
        }
    } else if ( compareValue > 0 ) { ////Recursive right subtree, continue to find the insertion position
        if ( node.right == null ) {
                node.right = new Node( newData, null , null );
            } else {
                this.insert( node.right, newData);
            }
        }
    }
}



class Node{
    constructor( data, left, right){
        this.data = data;
        this.left = left;
        this.right = right;
    }

    getData(){
        return this.data;
    }
    setData( data){
        this.data = data;
    }
}

////////////////////testing//////////////////

var binaryTree= new BinaryTree();

//Constructing a binary search tree
binaryTree.insert( binaryTree.getRoot(), 60 );
```

```
    binaryTree.insert( binaryTree.getRoot(), 40 );
    binaryTree.insert( binaryTree.getRoot(), 20 );
    binaryTree.insert( binaryTree.getRoot(), 10 );
    binaryTree.insert( binaryTree.getRoot(), 30 );
    binaryTree.insert( binaryTree.getRoot(), 50 );
    binaryTree.insert( binaryTree.getRoot(), 80 );
    binaryTree.insert( binaryTree.getRoot(), 70 );
    binaryTree.insert( binaryTree.getRoot(), 90 );

    document.write( "<br>delete node is:  10 <br>" );
    binaryTree.remove( binaryTree.getRoot(), 10 );

    document.write( "<br>In-order traversal binary tree <br>" );
    binaryTree.inOrder( binaryTree.getRoot());

    document.write( "<br>------------------------------------------<br>" );

    document.write( "<br>delete node is:  20<br>" );
    binaryTree.remove( binaryTree.getRoot(), 20 );

    document.write( "<br>In-order traversal binary tree<br>" );
    binaryTree.inOrder( binaryTree.getRoot());

    document.write( "<br>------------------------------------------<br>" );

    document.write( "<br>delete node is:  40<br>" );
    binaryTree.remove( binaryTree.getRoot(), 40 );

    document.write( "<br>In-order traversal binary tree<br>" );
    binaryTree.inOrder( binaryTree.getRoot());
</script>
```

**Result:**


TestBinarySearchTreeDelete.ht
file:///F:/TestBinarySearchTreeDelete.html

```
delete node is: 10

In-order traversal binary tree
20, 30, 40, 50, 60, 70, 80, 90,
------------------------------------------------

delete node is: 20

In-order traversal binary tree
30, 40, 50, 60, 70, 80, 90,
------------------------------------------------

delete node is: 40

In-order traversal binary tree
30, 50, 60, 70, 80, 90,
```

# Binary Heap Sorting

**Binary Heap Sorting:**
The value of the non-terminal node in the binary tree is not greater than the value of its left and right child nodes.

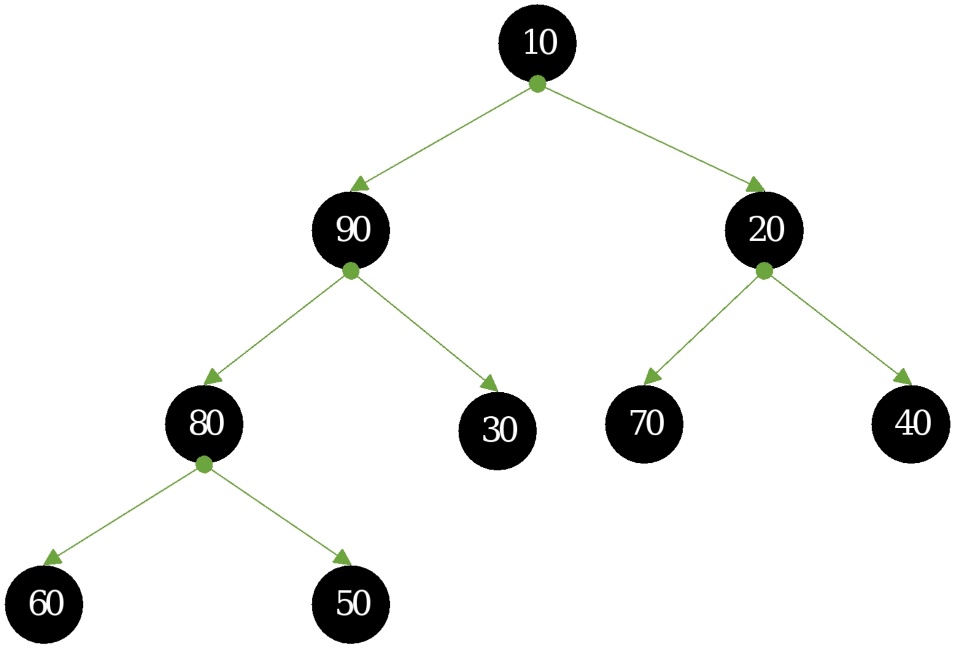
Small top heap : ki <= k2i and ki <= k2i+1
Big top heap :ki >= k2i and ki >= k2i+1

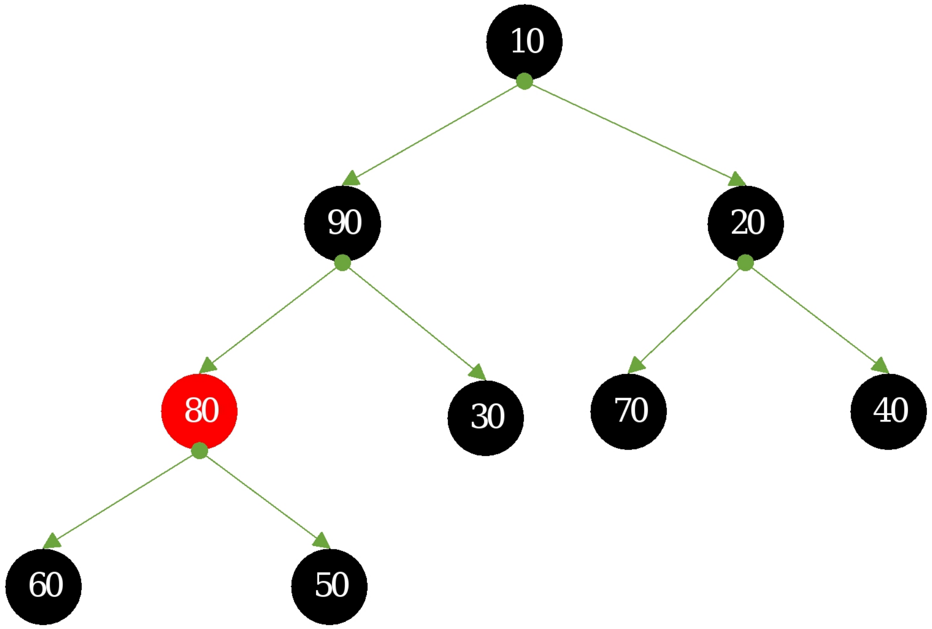
Parent node subscript = (i-1)/2
Left subnode subscript = 2*i+1
Right subnode subscript = 2*i+2

**Heap sorting process:**
1. Build a heap
2. After outputting the top element of the heap, adjust from top to bottom, compare the top element with the root node of its left and right subtrees, and swap the smallest element to the top of the heap; then adjust continuously until the leaf nodes to get new heap.

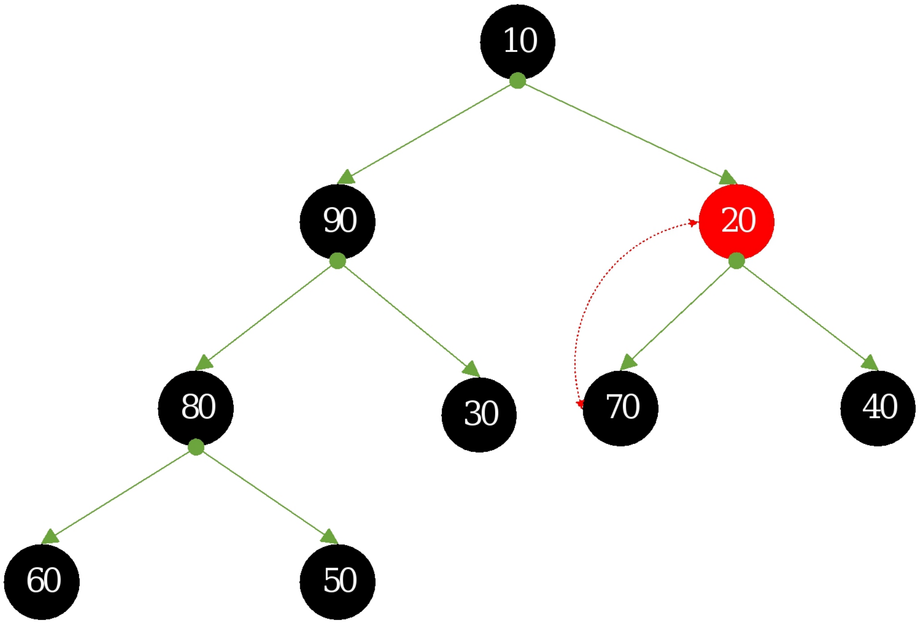
**1. {10, 90, 20, 80, 30, 70, 40, 60, 50} build heap and then heap sort output.**

**Initialize the heap and build the heap**

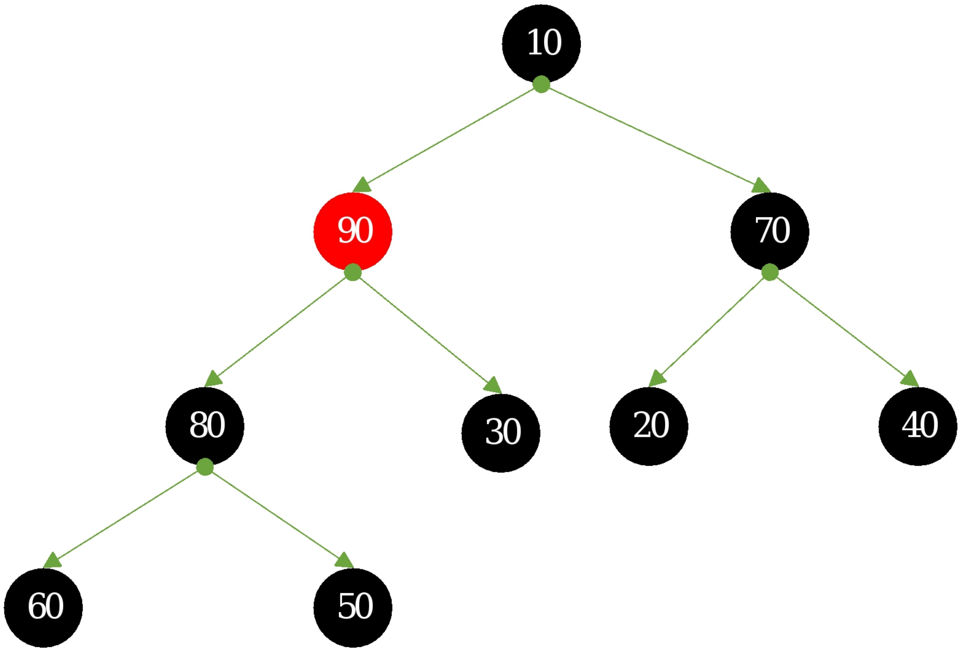

10
90
20
80
30
70
40
60
50


10
90
20
80
30
70
40
60
50

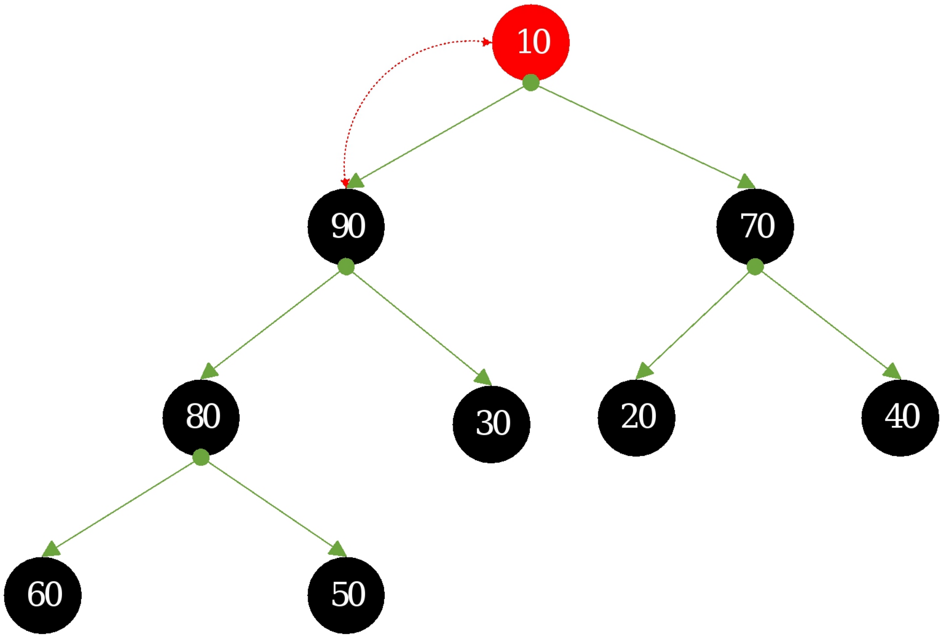
Not Leaf Node = 80 > left = 60 □ 80 > right = 50 No need to move


10
90
20
80
30
70
40
60
50

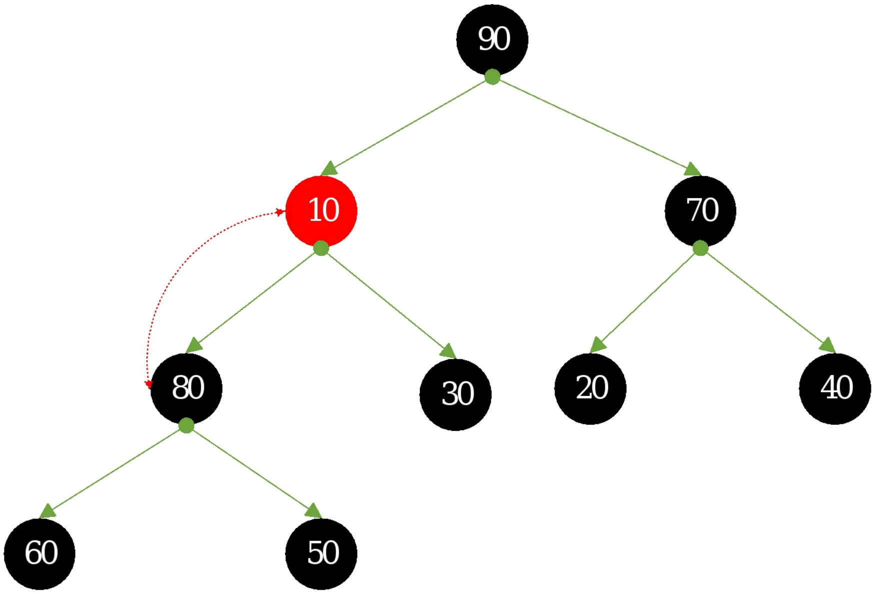
Not Leaf Node = 20 < left = 70 , 70 > right = 40 , 20 swap with 70


10
90
70
80
30
20
40
60
50

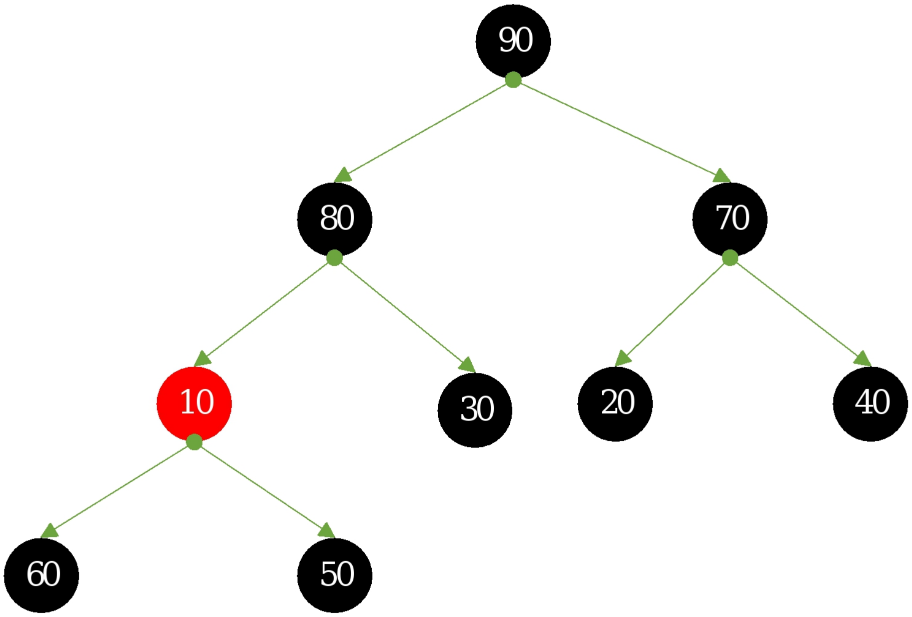
Not Leaf Node =90 >left =80□ 80 >right =30 No need to move


10
90
70
80
30
20
40
60
50

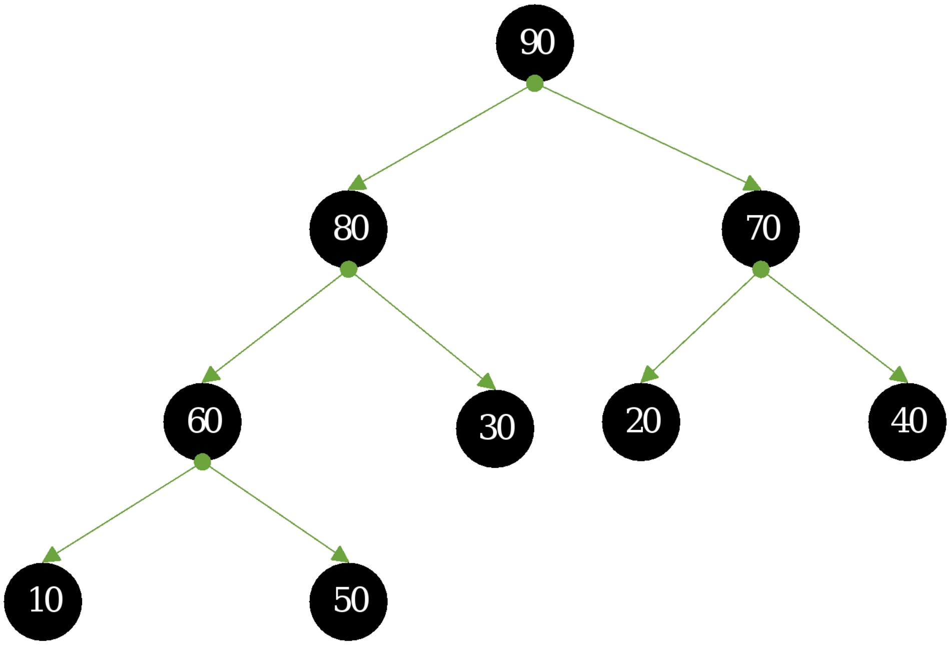
Not Leaf Node =10 <left =90 , 90 >right =70 , 10 swap with 90


90
10
70
80
30
20
40
60
50

Still Not Leaf Node =10 <left =80 , 80 >right =30 , 10 swap with 80

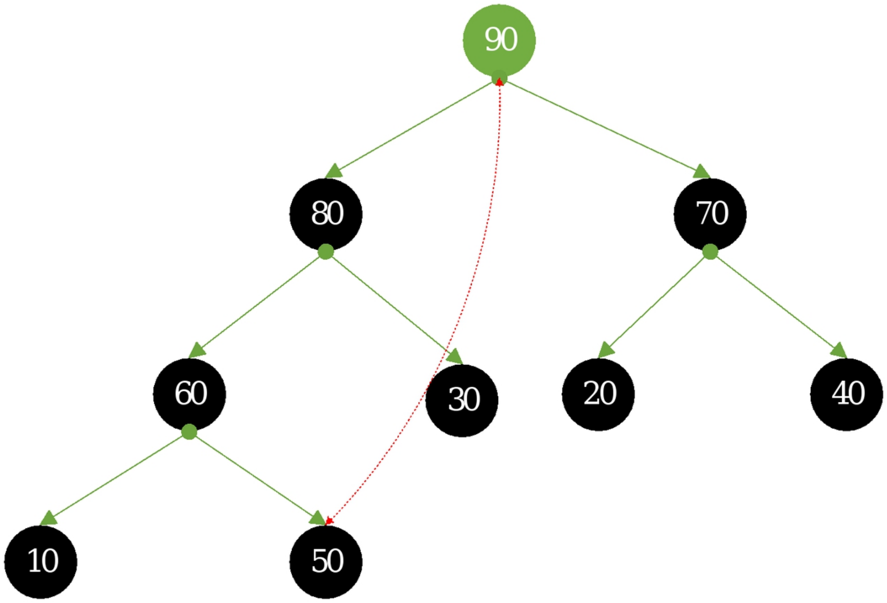

90
80
70
10
30
20
40
60
50

Still Not Leaf Node =10 <left =60 , 60 >right =50 , 10 swap with 60

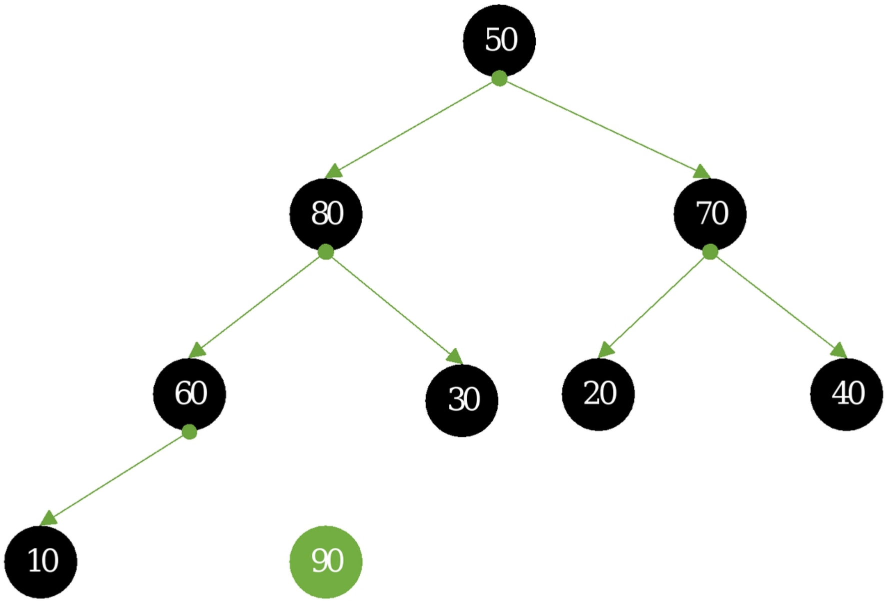

90
80
70
60
30
20
40
10
50

Complete construction of the heap

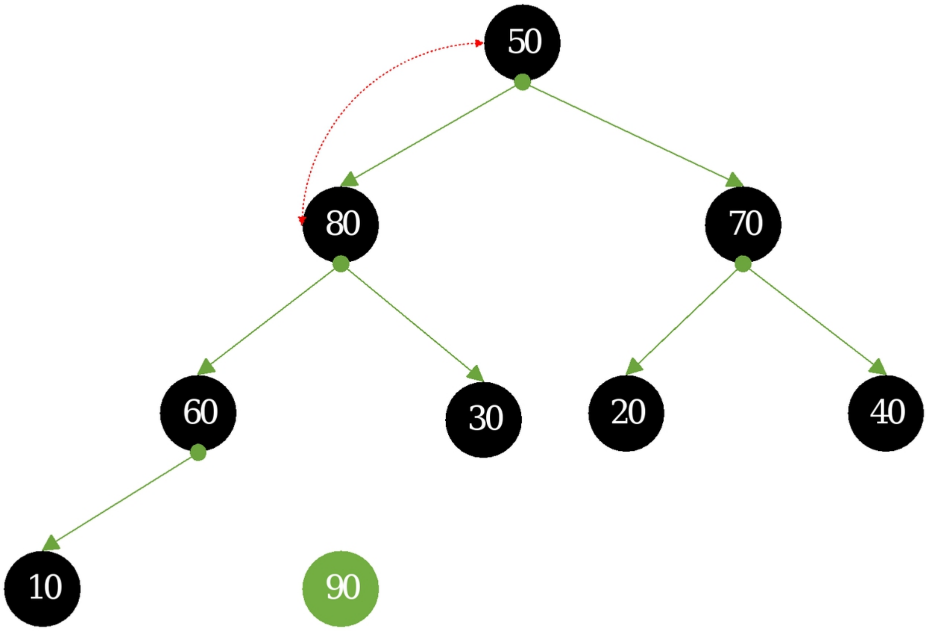
## 2. Start heap sorting


90
80
70
60
30
20
40
10
50

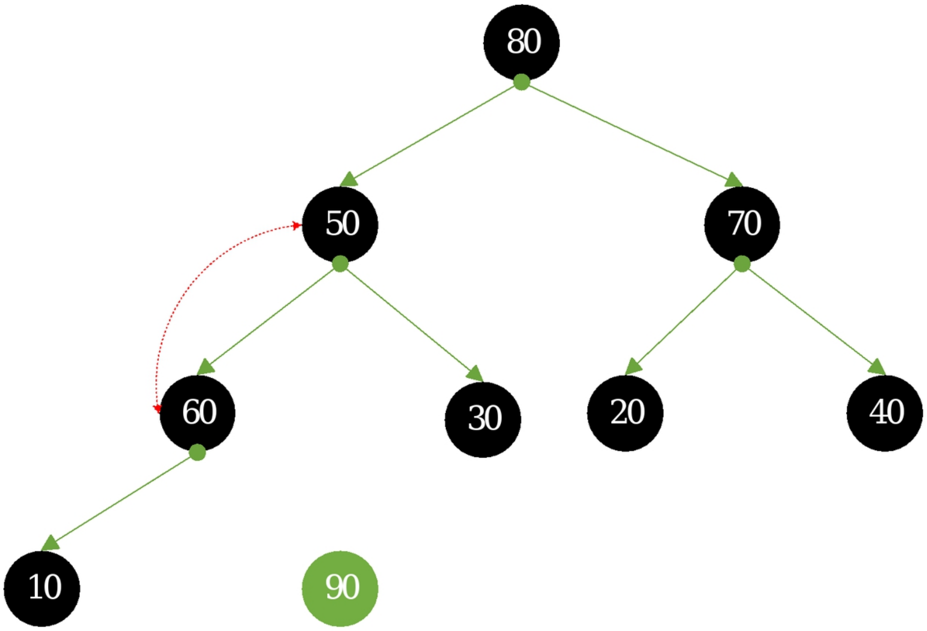
root = 90 and tail = 50 are exchanged


50
80
70
60
30
20
40
10
90

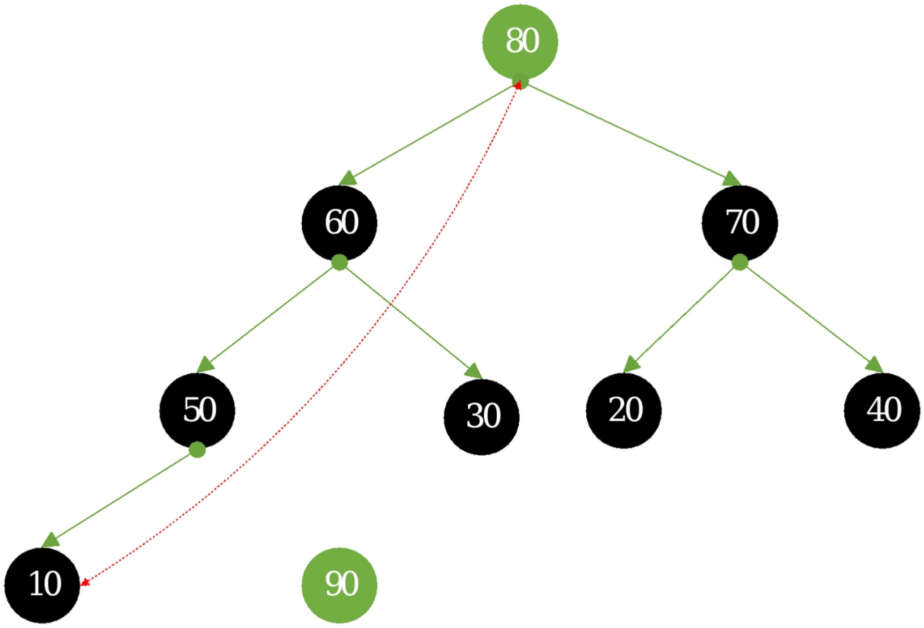
# adjust the heap



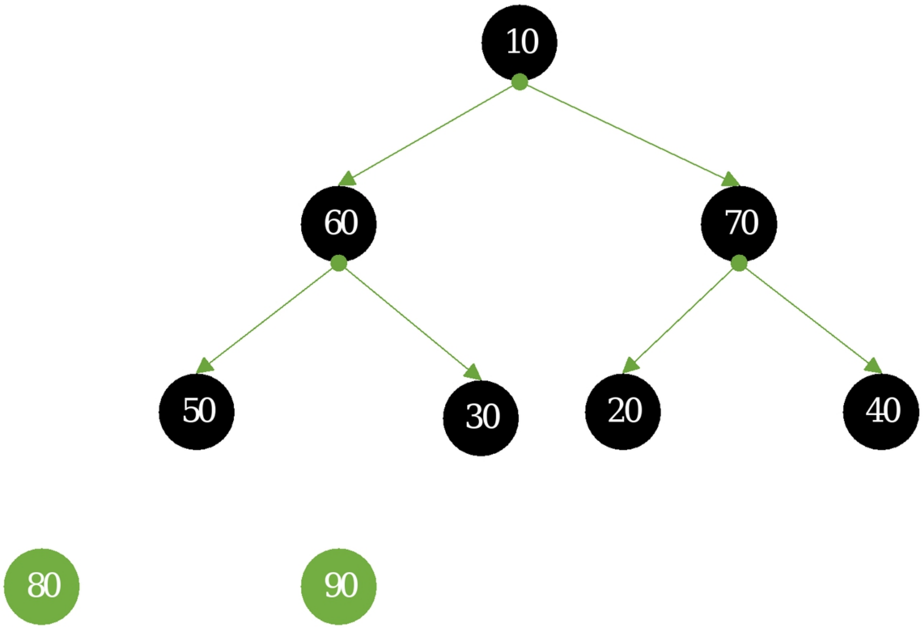
50
80
70
60
30
20
40
10
90



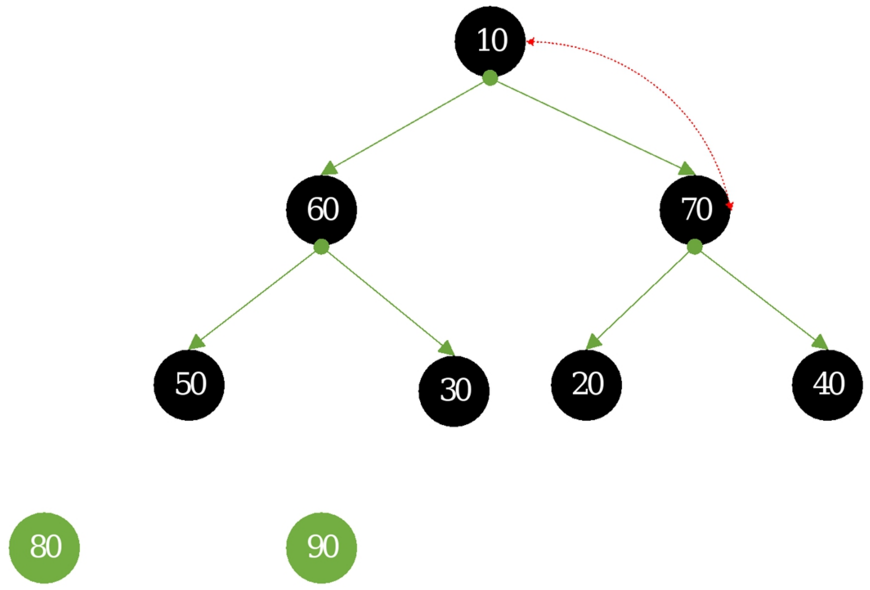
80
50
70
60
30
20
40
10
90

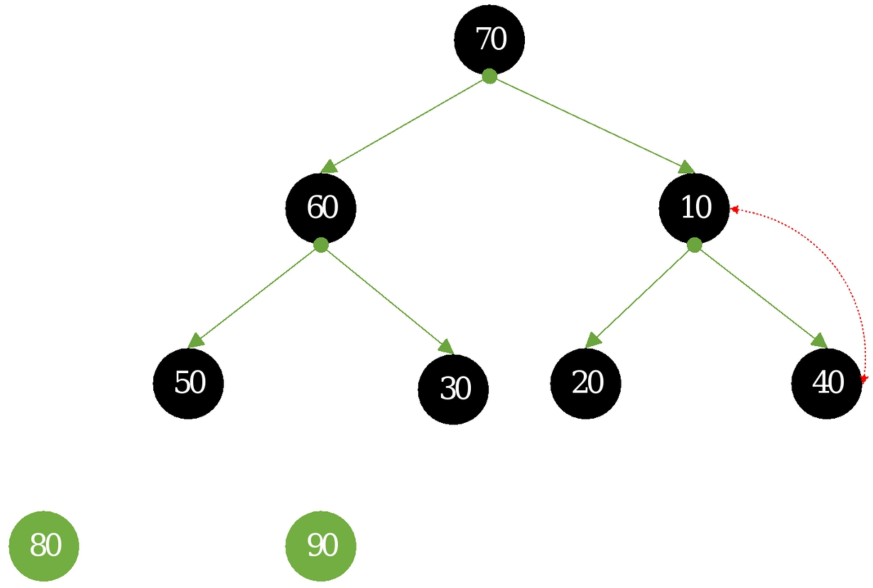

80
60
70
50
30
20
40
10
90

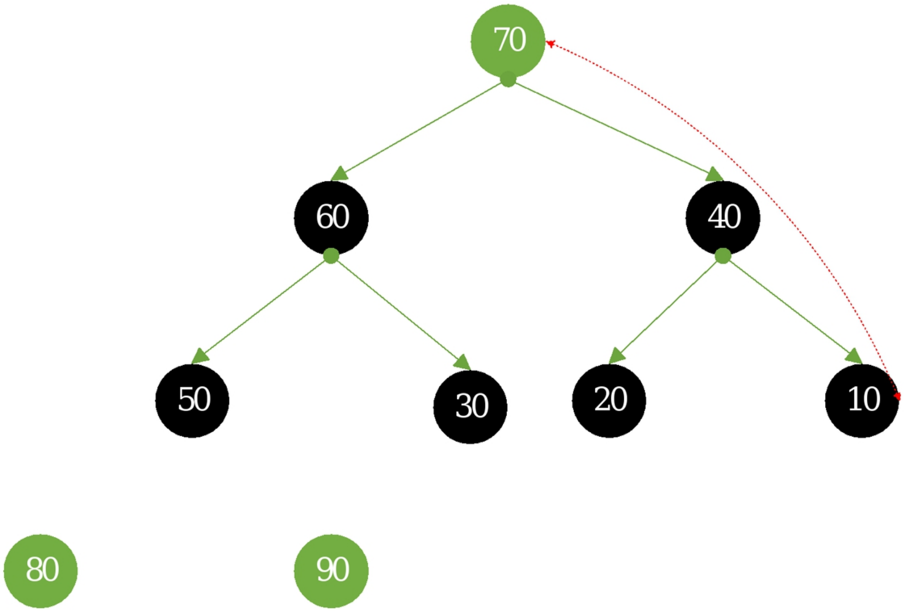
root =80 and tail =10 are exchanged


10
60
70
50
30
20
40
80
90

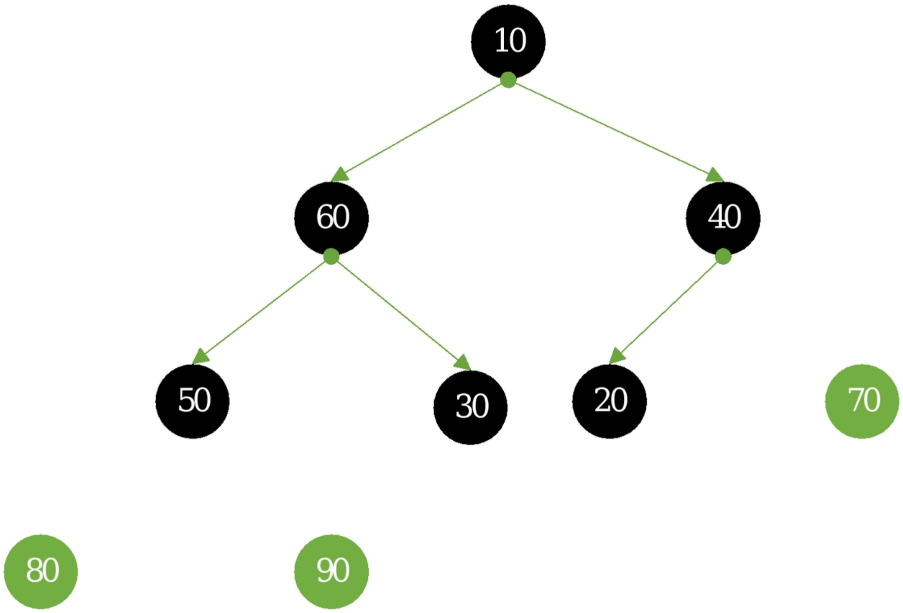
# adjust the heap


10
60
70
50
30
20
40
80
90

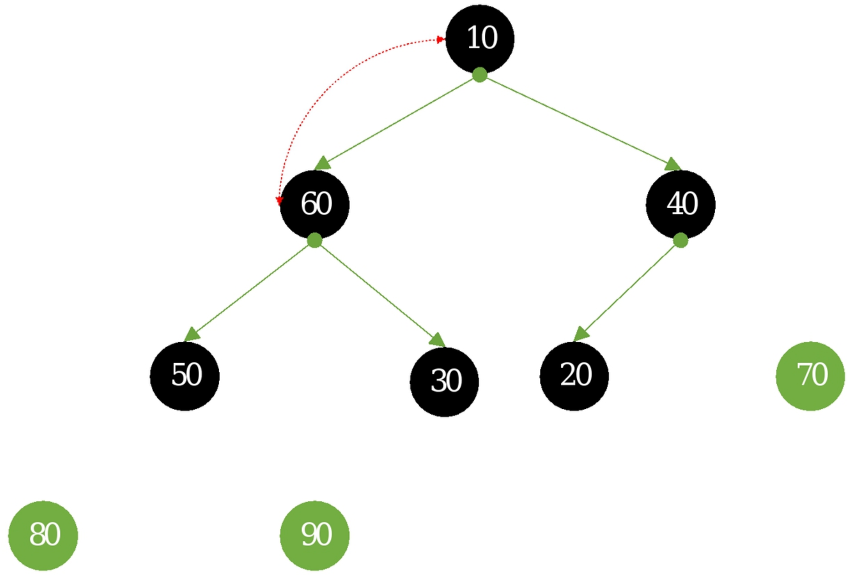

70
60
10
50
30
20
40
80
90

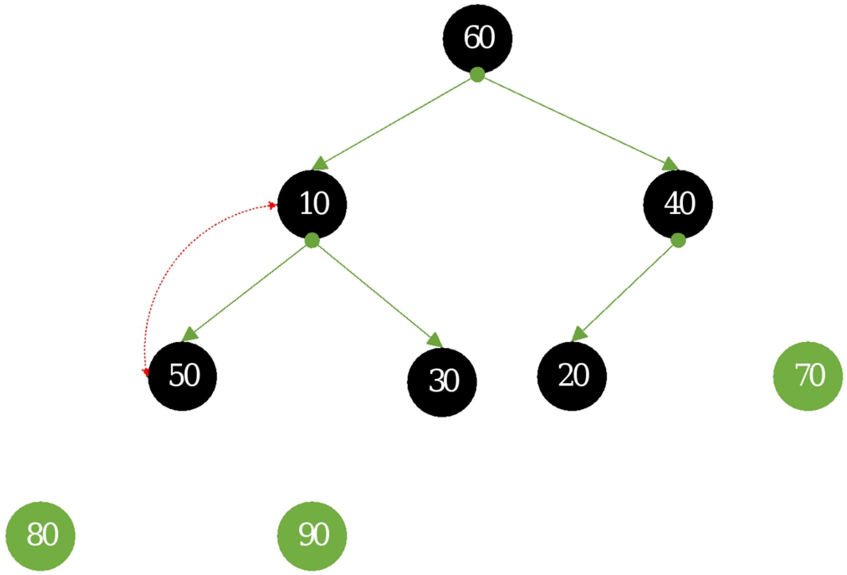

70
60
40
50
30
20
10
80
90

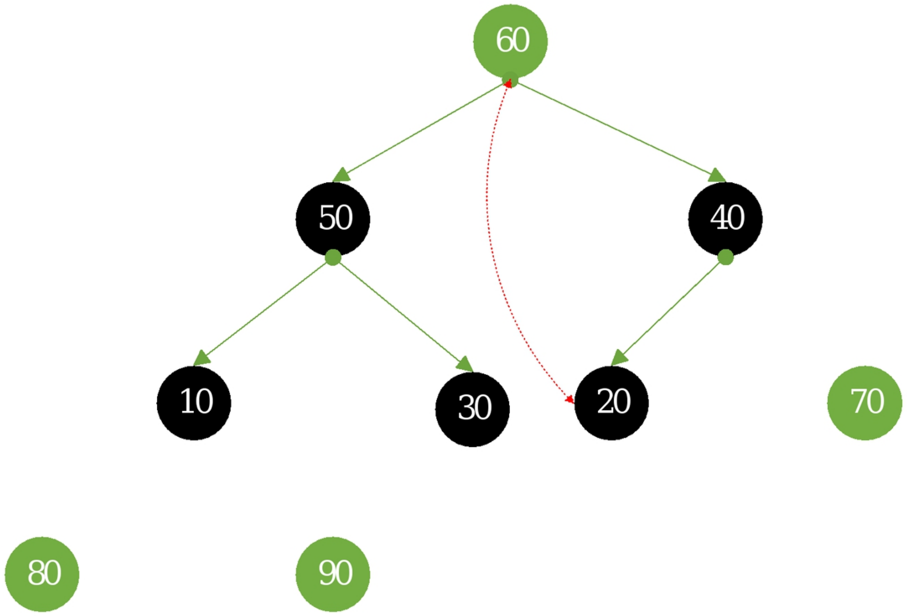
root =70 and tail =10 are exchanged


10
60
40
50
30
20
70
80
90

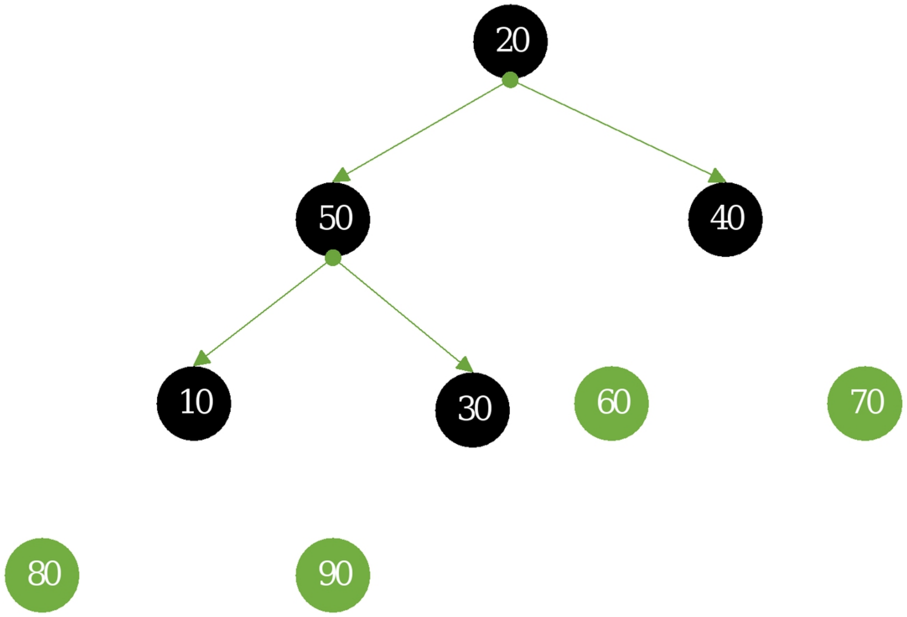
# adjust the heap


10
60
40
50
30
20
70
80
90

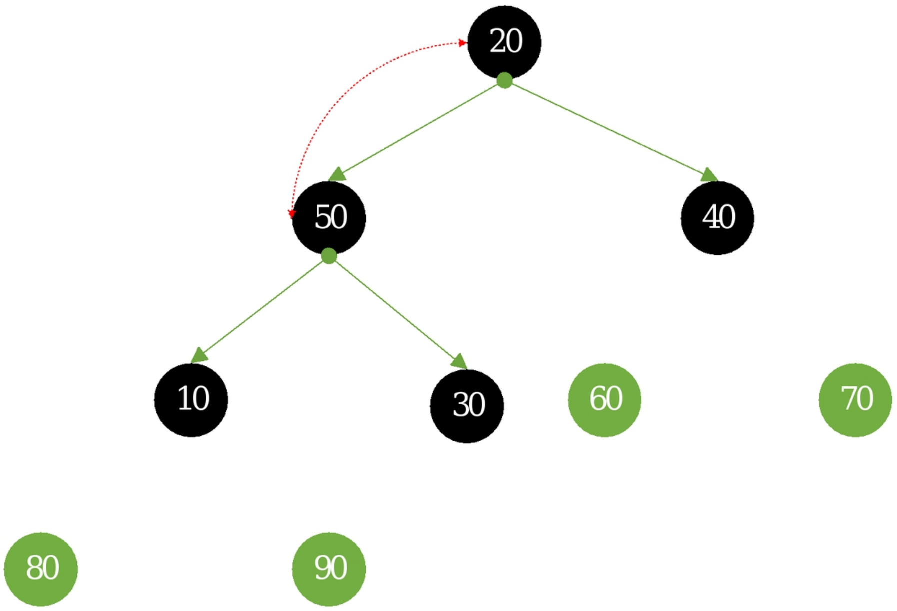

60
10
40
50
30
20
70
80
90

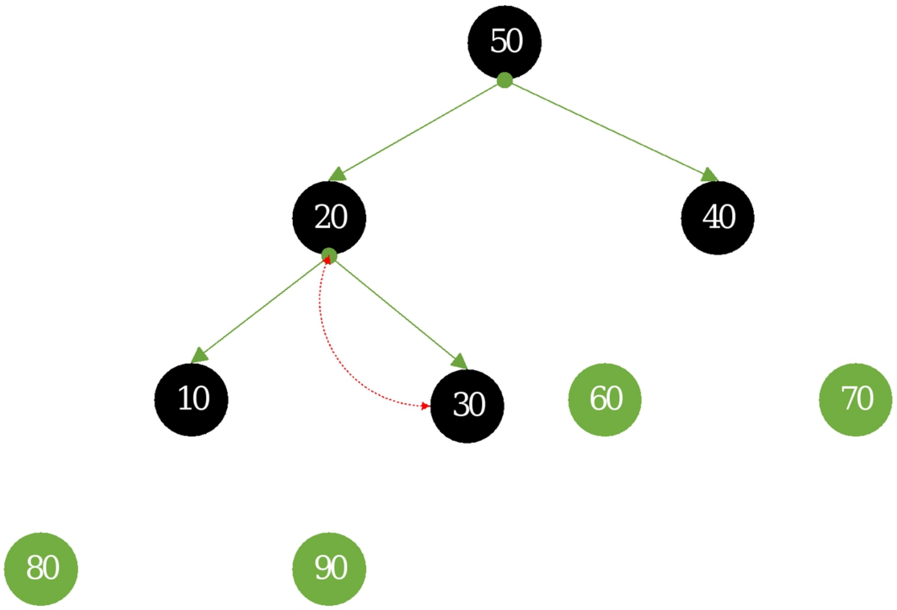

60
50
40
10
30
20
70
80
90

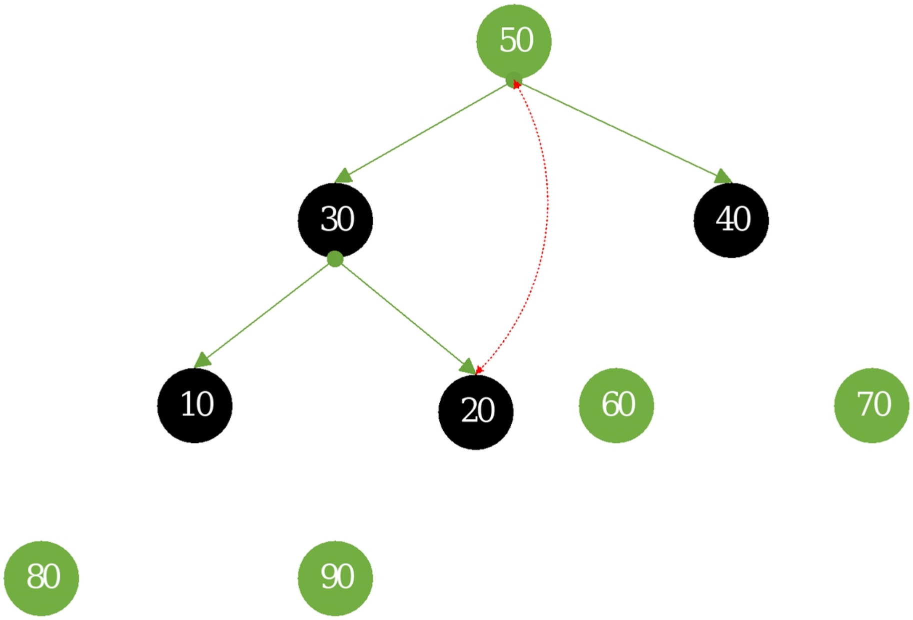
root = 60 and tail = 20 are exchanged


20
50
40
10
30
60
70
80
90

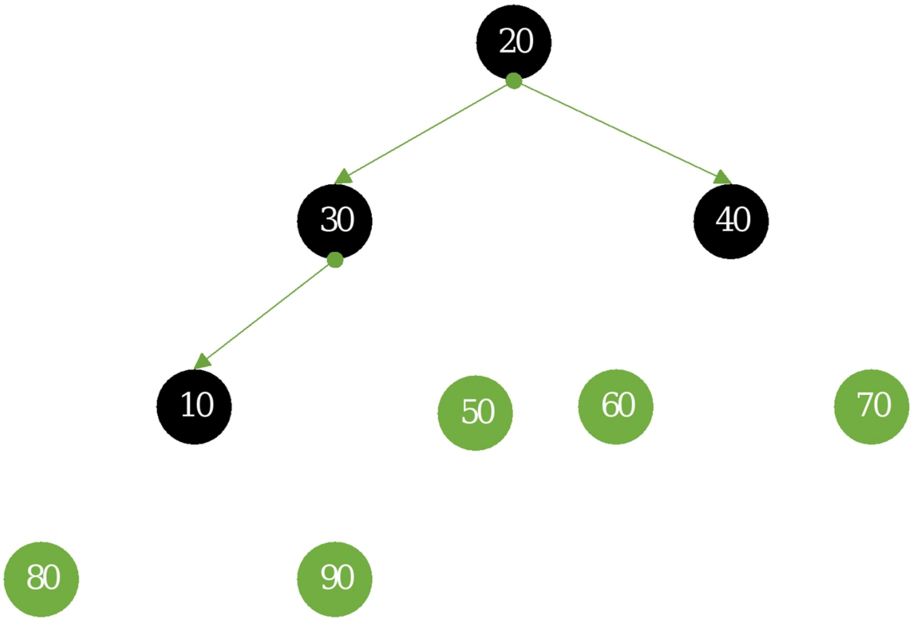
# adjust the heap


20
50
40
10
30
60
70
80
90

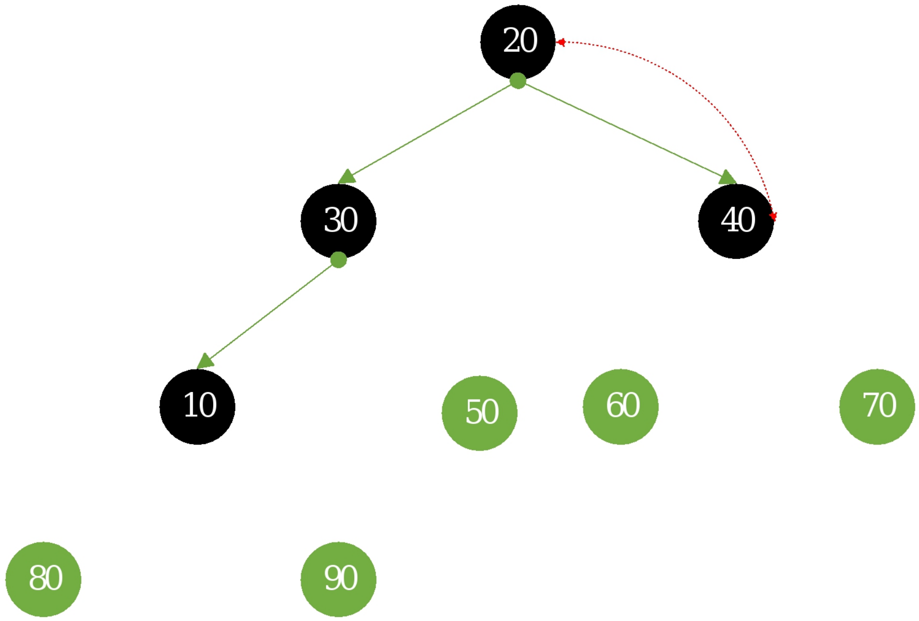

50
20
40
10
30
60
70
80
90

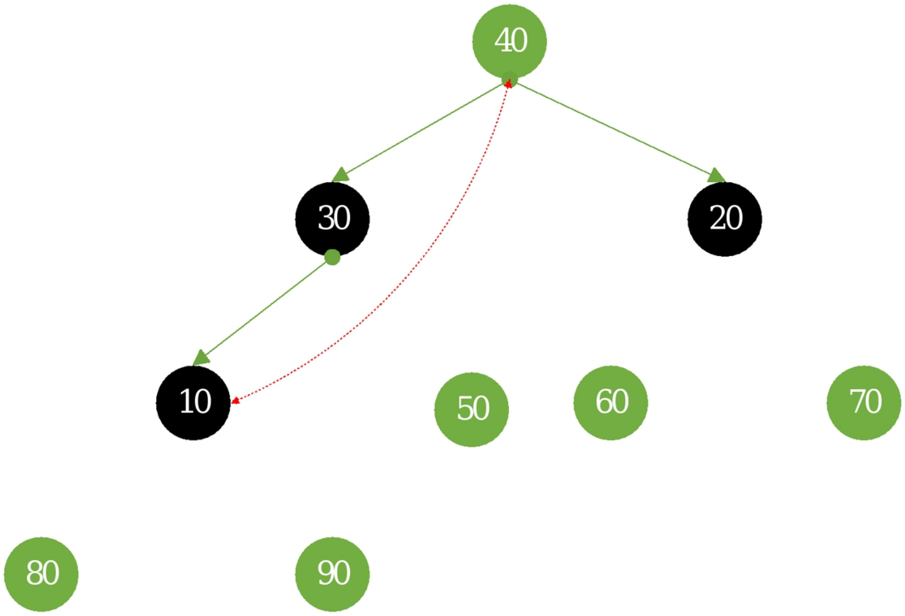

50
30
40
10
20
60
70
80
90

root = 50 and tail = 20 are exchanged

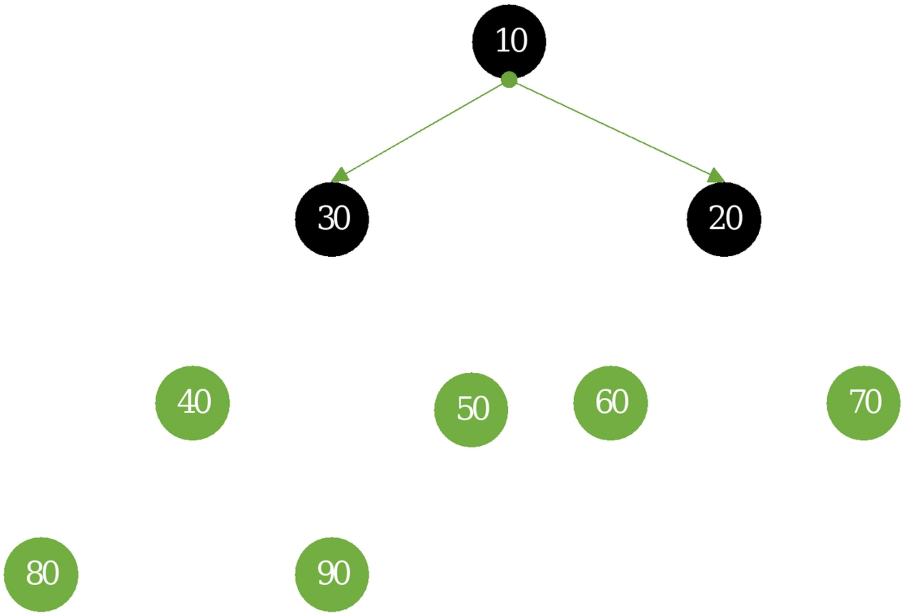

20
30
40
10
50
60
70
80
90

# adjust the heap

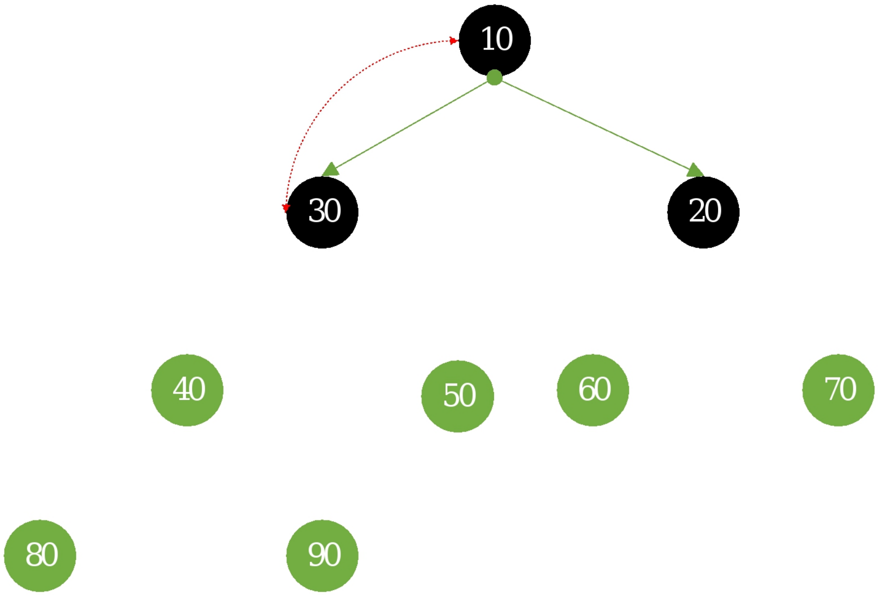

20
30
40
10
50
60
70
80
90

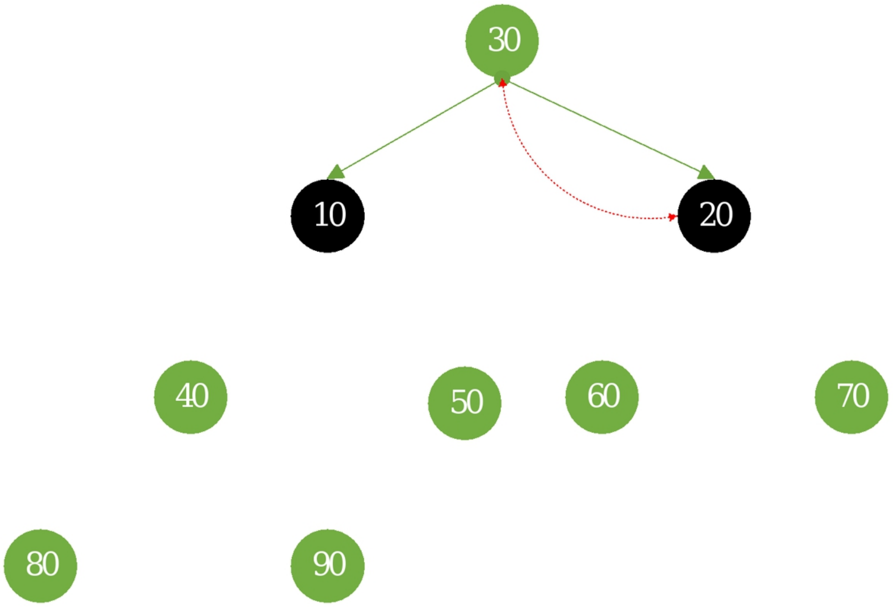

40
30
20
10
50
60
70
80
90

root =40 and tail =10 are exchanged

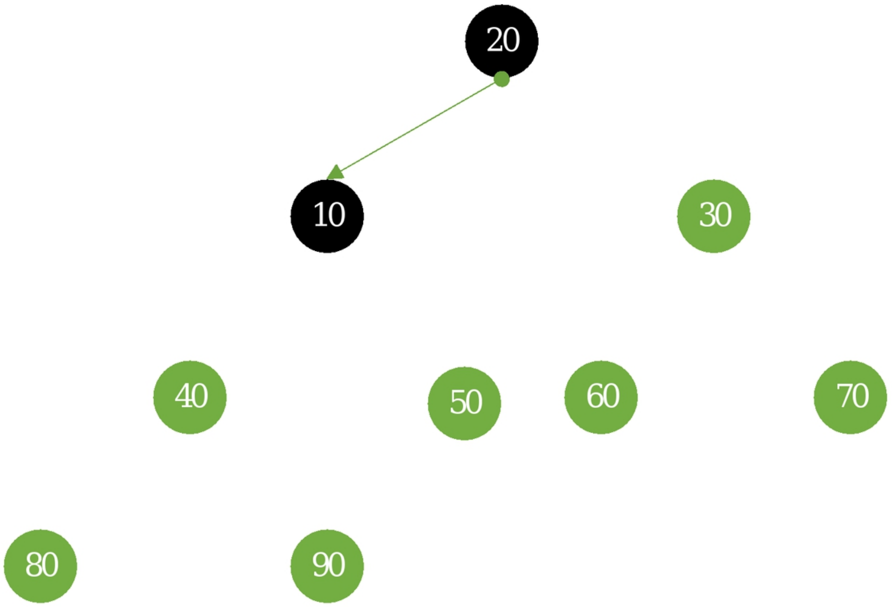

10
30
20
40
50
60
70
80
90

# adjust the heap

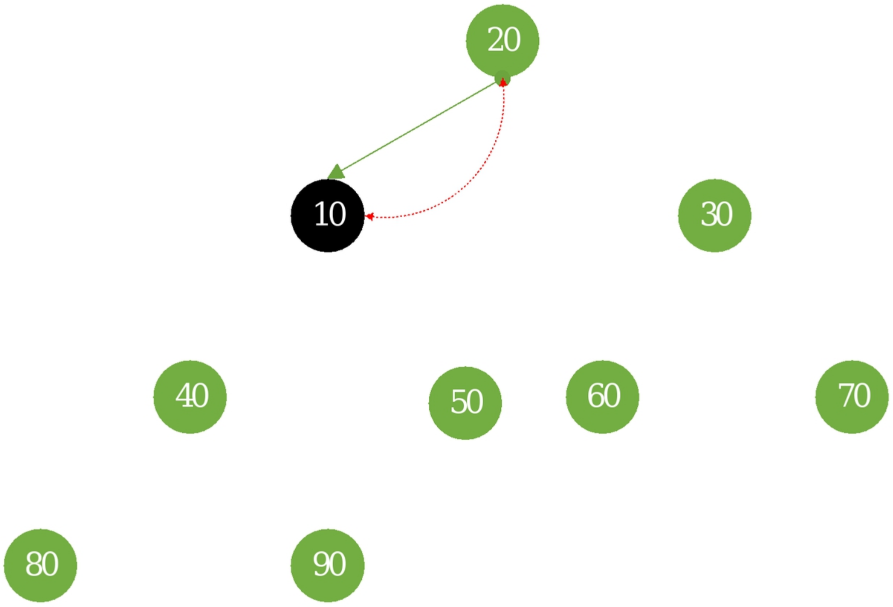

10
30
20
40
50
60
70
80
90


30
10
20
40
50
60
70
80
90

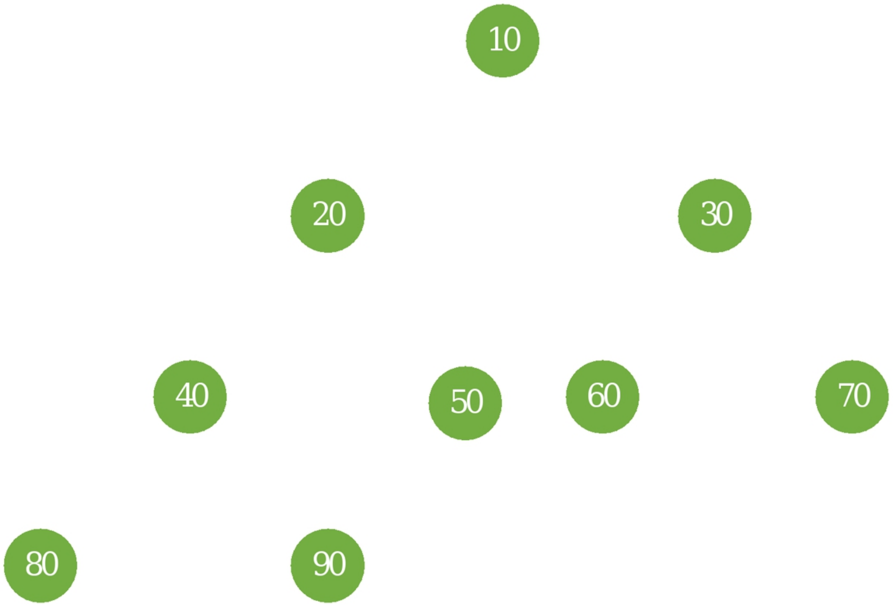
root =30 and tail =20 are exchanged


20
10
30
40
50
60
70
80
90

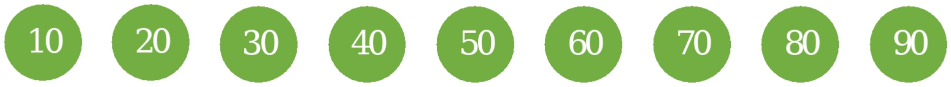
no need adjust the heap


20
10
30
40
50
60
70
80
90

root =20 and tail =10 are exchanged


10
20
30
40
50
60
70
80
90

### Heap sort result

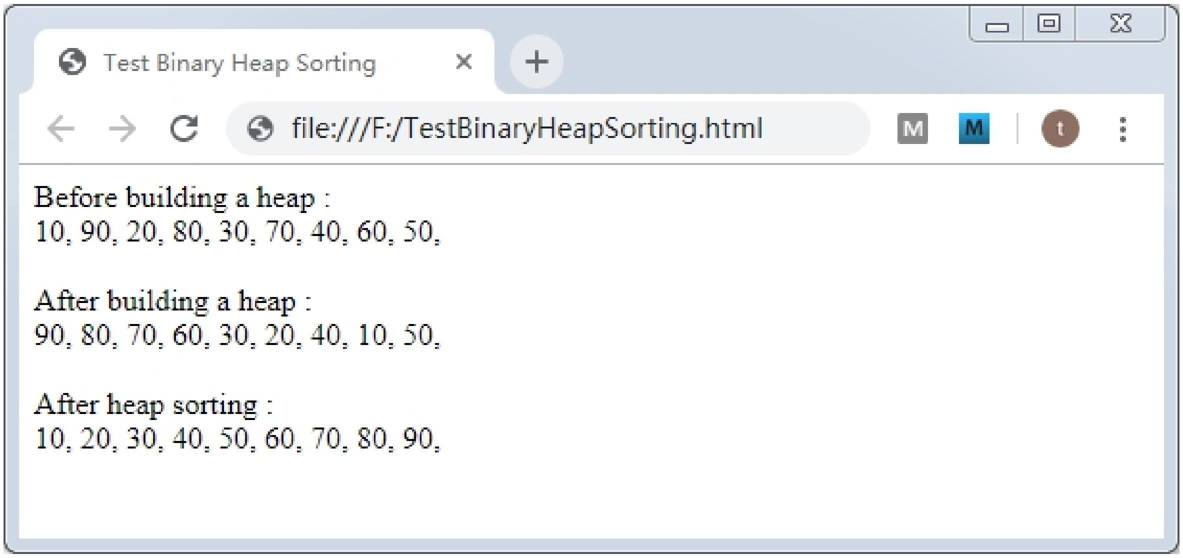

10
20
30
40
50
60
70
80
90

**Create a TestBinaryHeapSorting.html with Notepad**

```
<script type ="text/javascript" >
    class HeapSort {
        constructor(){
            this.array = [];
        }

        //Initialize the heap
        createHeap( array) {
            this.array = array;

            // Build a heap, (array.length - 1) / 2 scan half of the nodes with child nodes
            for ( var i = ( array.length - 1 ) / 2 ; i >= 0 ; i--) {
                this.adjustHeap( i, array.length - 1 );
            }
        }

        //Adjustment heap
        adjustHeap( currentIndex, maxLength) {
            var noLeafValue = this.array[ currentIndex]; // Current non-leaf node

            //2 * currentIndex + 1  Current left subtree subscript
            for ( var j = 2 * currentIndex + 1 ; j <= maxLength; j = currentIndex * 2 + 1 ) {
                if ( j < maxLength && this.array[ j] < this.array[ j + 1 ]) {
                    j++; // j Large subscript
                }

                if ( noLeafValue >= this.array[ j]) {
                    break ;
                }

                this.array[ currentIndex] = this.array[ j]; // Move up to the parent node
                currentIndex = j;
        }

        this.array[ currentIndex] = noLeafValue; // To put in the position
    }


    heapSort() {
        for ( var i = this.array.length - 1 ; i > 0 ; i--) {
            var temp = this.array[ 0 ];
            this.array[ 0 ] = this.array[ i];
            this.array[ i] = temp;
            this.adjustHeap( 0 , i - 1 );
        }
    }
}

//////////////////////testing////////////////////

var heapSort = new HeapSort();

var scores = [ 10 , 90 , 20 , 80 , 30 , 70 , 40 , 60 , 50 ];

document.write( "Before building a heap : <br>" );
for ( var i = 0 ; i < scores.length; i++) {
    document.write( scores[ i] + ", " );
}
document.write( "<br><br>" );

/////////////////////////////////////////////////////////////

document.write( "After building a heap : <br>" );
heapSort.createHeap( scores);
for ( var i = 0 ; i < scores.length; i++) {
    document.write( scores[ i] + ", " );
}
document.write( "<br><br>" );
```

```
    ////////////////////////////////////////////////////////////

    document.write( "After heap sorting : <br>" );
    heapSort.heapSort();
    for ( var i = 0 ; i < scores.length; i++) {
        document.write( scores[ i] + ", " );
    }

</script>
```

**Result:**

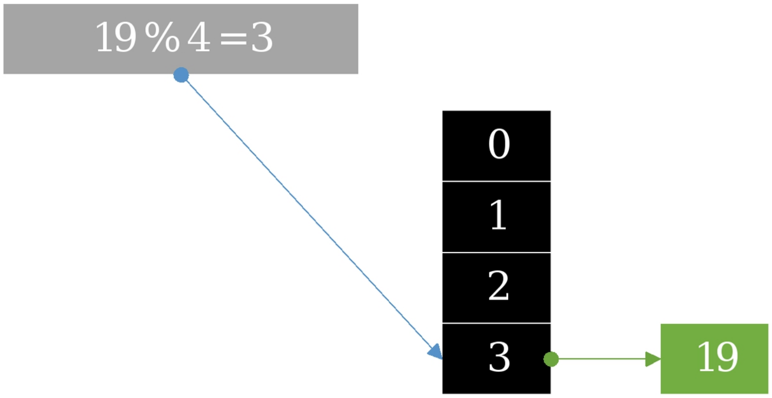

Test Binary Heap Sorting
file:///F:/TestBinaryHeapSorting.html

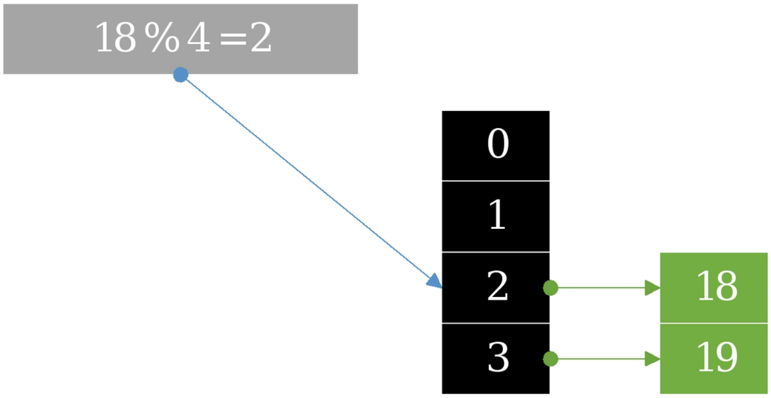
```
Before building a heap :
10, 90, 20, 80, 30, 70, 40, 60, 50,

After building a heap :
90, 80, 70, 60, 30, 20, 40, 10, 50,

After heap sorting :
10, 20, 30, 40, 50, 60, 70, 80, 90,
```

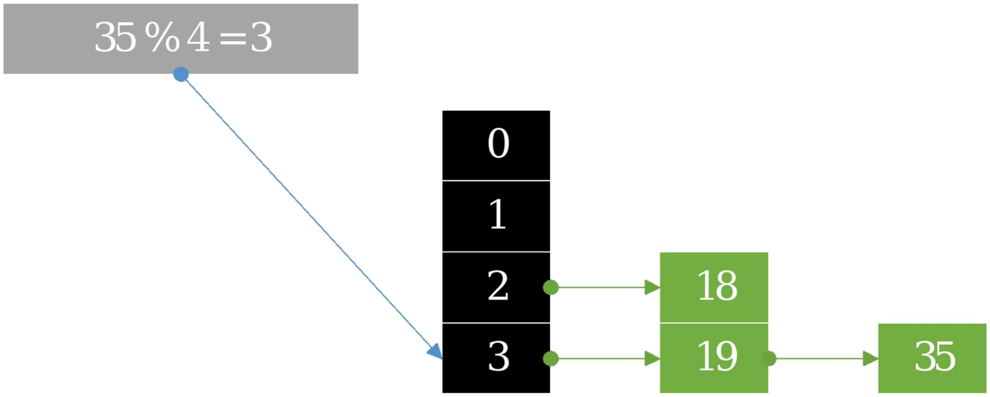
# Hash Table

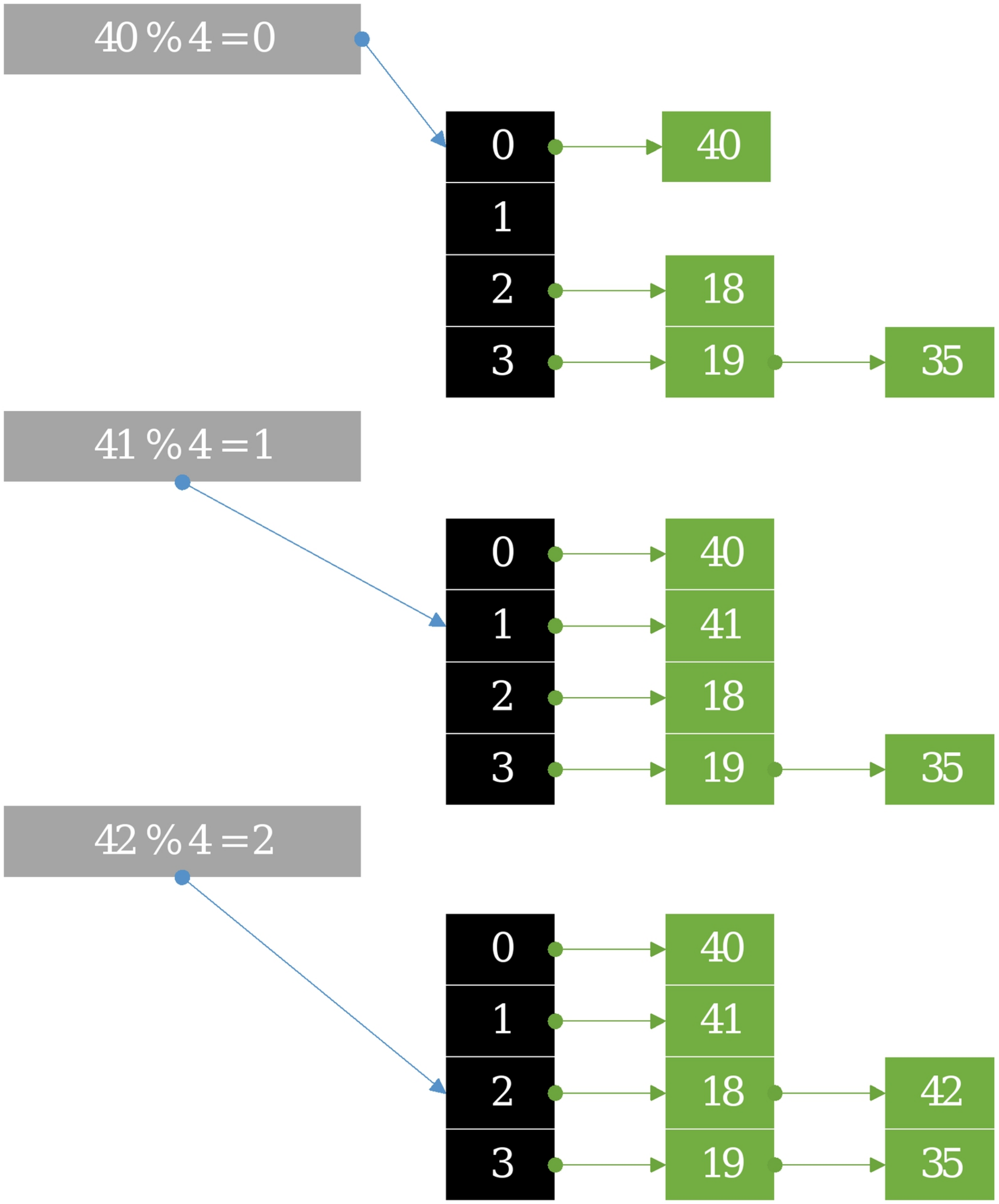
**Hash Table:**
Access by mapping key => values in the table.

**1. Map {19, 18, 35,40,41,42} to the HashTable mapping rule key % 4**

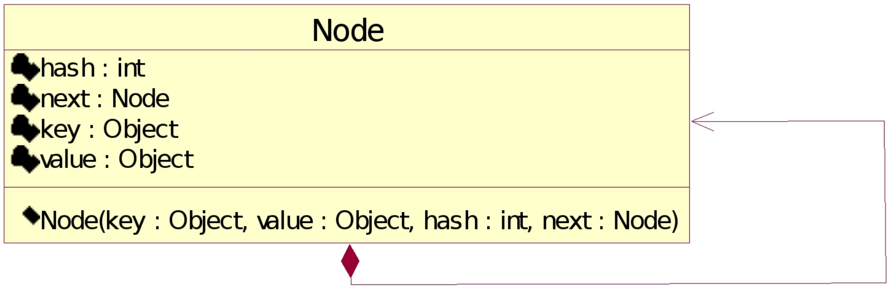

19 % 4 =3
0
1
2
3
19


18 % 4 =2
0
1
2
3
18
19


35 % 4 =3
0
1
2
3
18
19
35


40 % 4 = 0
41 % 4 = 1
42 % 4 = 2

## 2. Implement a Hashtable


Node
hash : int
next : Node
key : Object
value : Object
Node(key : Object, value : Object, hash : int, next : Node)

### Create a TestHashtable.html with Notepad

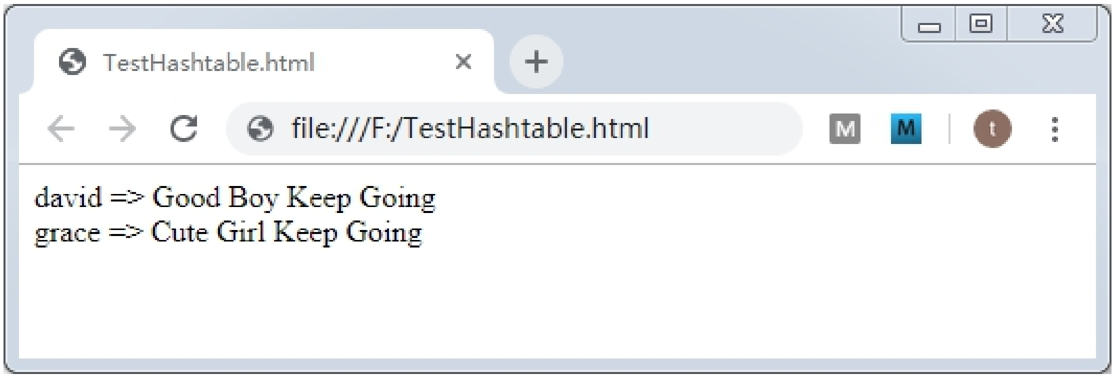
```
<script type ="text/javascript" >
   class Node {
      constructor( key, value, hash, next){
         this.key = key;
         this.value = value;
         this.hash = hash;
         this.next = next;
      }

      getKey() {
         return this.key;
      }
      setKey( key) {
         this.key = key;
      }

      getValue() {
         return this.value;
      }
      setValue( value) {
         this.value = value;
      }

      getHash() {
         return this.hash;
      }
```

```
    setHash( hash) {
      this.hash = hash;
    }
  }

  class Hashtable {
    constructor(){
      this.capacity = 16 ;
      this.table = new Array( this.capacity);
      this.size = 0 ;
    }

    size() {
      return this.size;
    }

    isEmpty() {
      return this.size == 0 ? true : false ;
    }

    hash( key) {
      var hash = 0 ;
      for ( var i = 0 ; i < key.length; i++) {
        hash += key.charCodeAt( i);
      }
      return hash % 37 ;
    }

    //Calculate the hash value according to the key hash algorithm
    hashCode( key) {
      var avg = this.hash( key) * ( Math.pow( 5 , 0.5 ) - 1 ) / 2 //hash policy for middle-square method
      var numeric = avg - Math.floor( avg)
      return parseInt( Math.floor( numeric * this.table.length))
    }
```

```
put( key, value) {
    if ( key == null ) {
        throw new IllegalArgumentException();
    }

    var hash = this.hashCode( key);
    var newNode = new Node( key, value, hash, null );
    var node = table[ hash];
    while ( node != null ) {
        if ( node.getKey(). equals( key)) {
            node.setValue( value);
            return ;
        }
        node = node.next;
    }
    newNode.next = this.table[ hash];
    this.table[ hash] = newNode;
    this.size++;
}

get( key) {
    if ( key == null ) {
        return null ;
    }
```

```
            var hash = this.hashCode( key);
            var node = this.table[ hash];
            while ( node != null ) { //Get value according to key
                if ( node.getKey() == key) {
                    return node.getValue();
                }
                node = node.next;
            }
            return null ;
        }

    }










    ////////////////////testing//////////////////

    var table = new Hashtable();

    table.put( "david" , "Good Boy Keep Going" );
    table.put( "grace" , "Cute Girl Keep Going" );

    document.write( "david => " + table.get( "david" ) + "<br>" );
    document.write( "grace => " + table.get( "grace" ) + "<br>" );


</script>
```

**Result:**

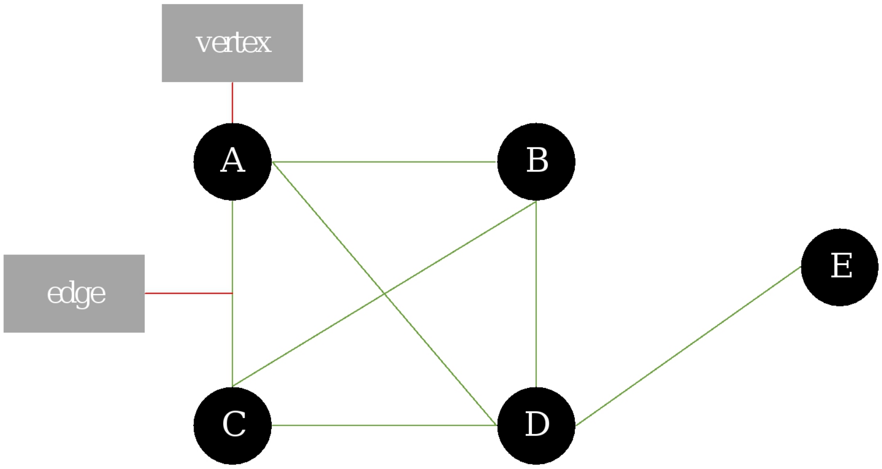

TestHashtable.html
file:///F:/TestHashtable.html

```
david => Good Boy Keep Going
grace => Cute Girl Keep Going
```

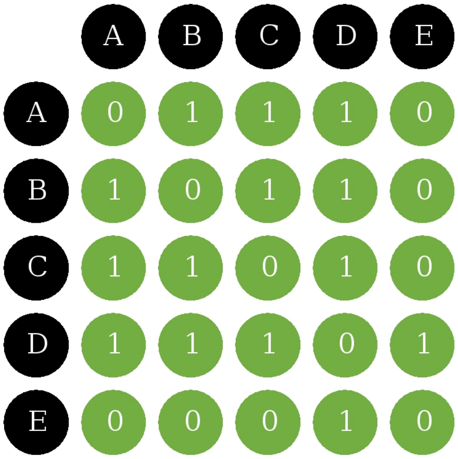
# Undirected Graph and Depth-Frst Search

**Undirected Graph:**
The data structure is represented by an adjacency matrix (that is, a two-dimensional array) and an adjacency list. Each node is called a vertex, and two adjacent nodes are called edges.

**Undirected Graph** No direction : A -> B = B -> A



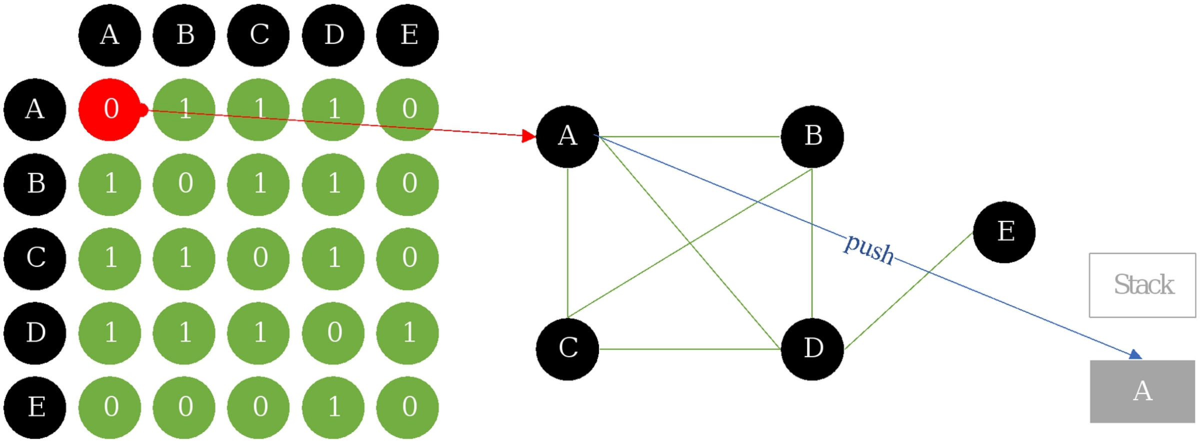
**1. The adjacency matrix is described above:**
The total number of vertices is a two-dimensional array size, if have value of the edge is 1 , otherwise no value of the edge is 0 .



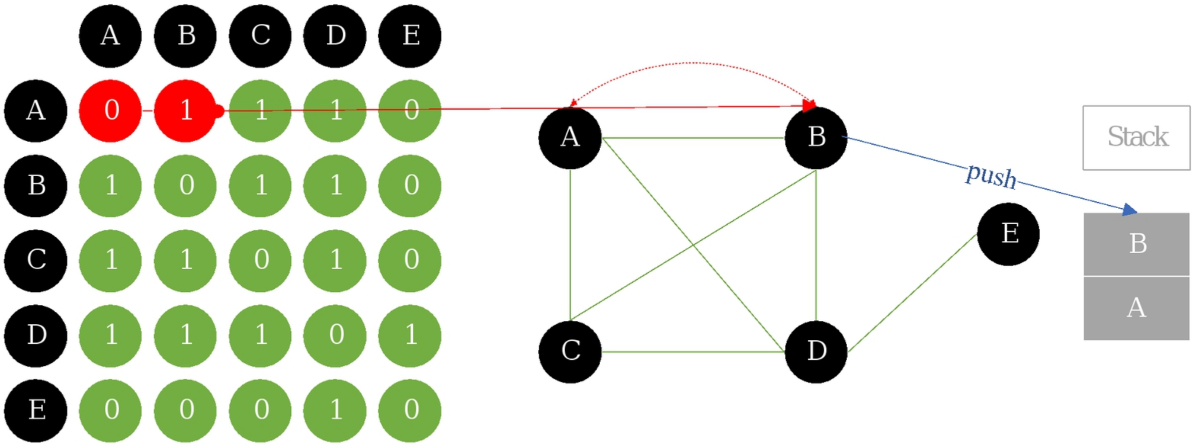
| | A | B | C | D | E |
|---|---|---|---|---|---|
| A | 0 | 1 | 1 | 1 | 0 |
| B | 1 | 0 | 1 | 1 | 0 |
| C | 1 | 1 | 0 | 1 | 0 |
| D | 1 | 1 | 1 | 0 | 1 |
| E | 0 | 0 | 0 | 1 | 0 |

## 2. Depth-Frst Search:

Look for the neighboring edge node B from A and then find the neighboring node C from B and so on until all nodes are found A -> B -> C -> D -> E .

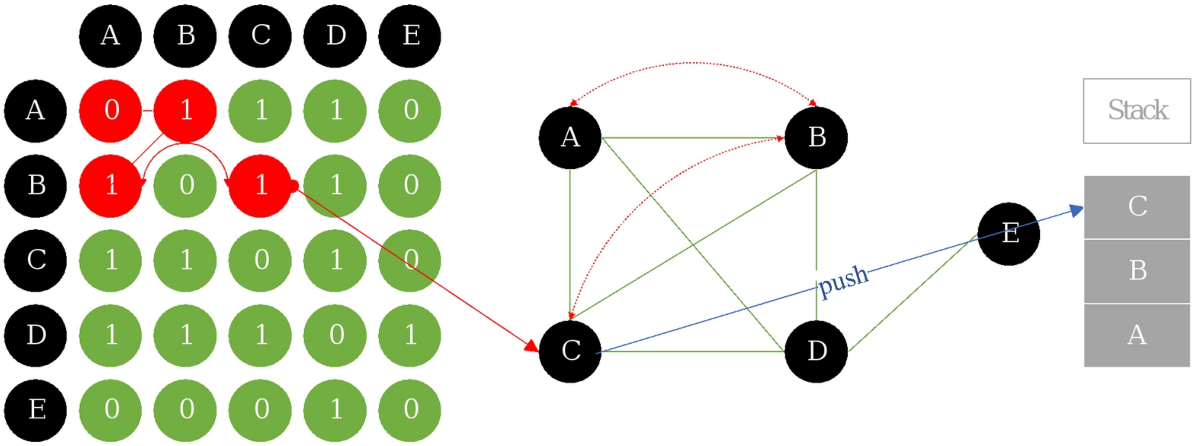


|   | A | B | C | D | E |
|---|---|---|---|---|---|
| A | 0 | 1 | 1 | 1 | 0 |
| B | 1 | 0 | 1 | 1 | 0 |
| C | 1 | 1 | 0 | 1 | 0 |
| D | 1 | 1 | 1 | 0 | 1 |
| E | 0 | 0 | 0 | 1 | 0 |



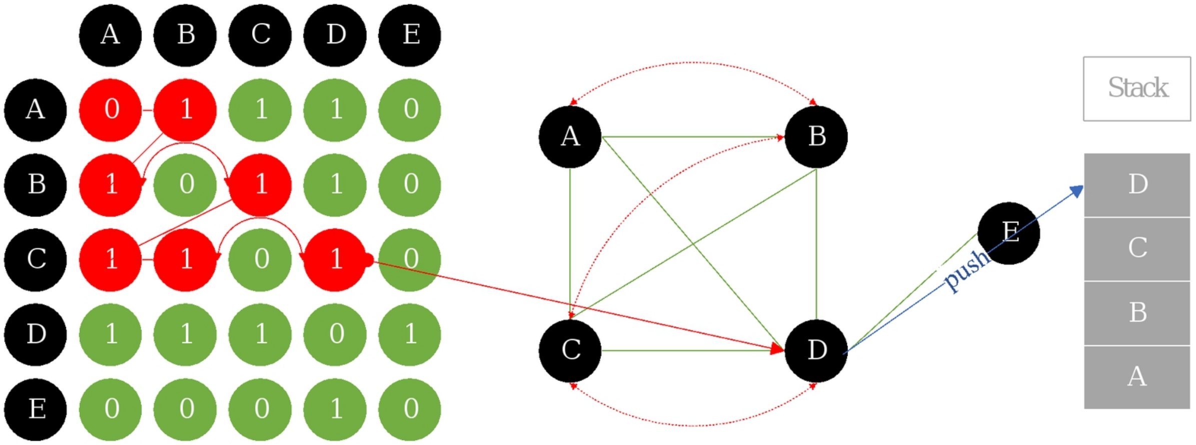
| | A | B | C | D | E |
|---|---|---|---|---|---|
| A | 0 | 1 | 1 | 1 | 0 |
| B | 1 | 0 | 1 | 1 | 0 |
| C | 1 | 1 | 0 | 1 | 0 |
| D | 1 | 1 | 1 | 0 | 1 |
| E | 0 | 0 | 0 | 1 | 0 |

A
B
C
D
E
push
Stack
B
A



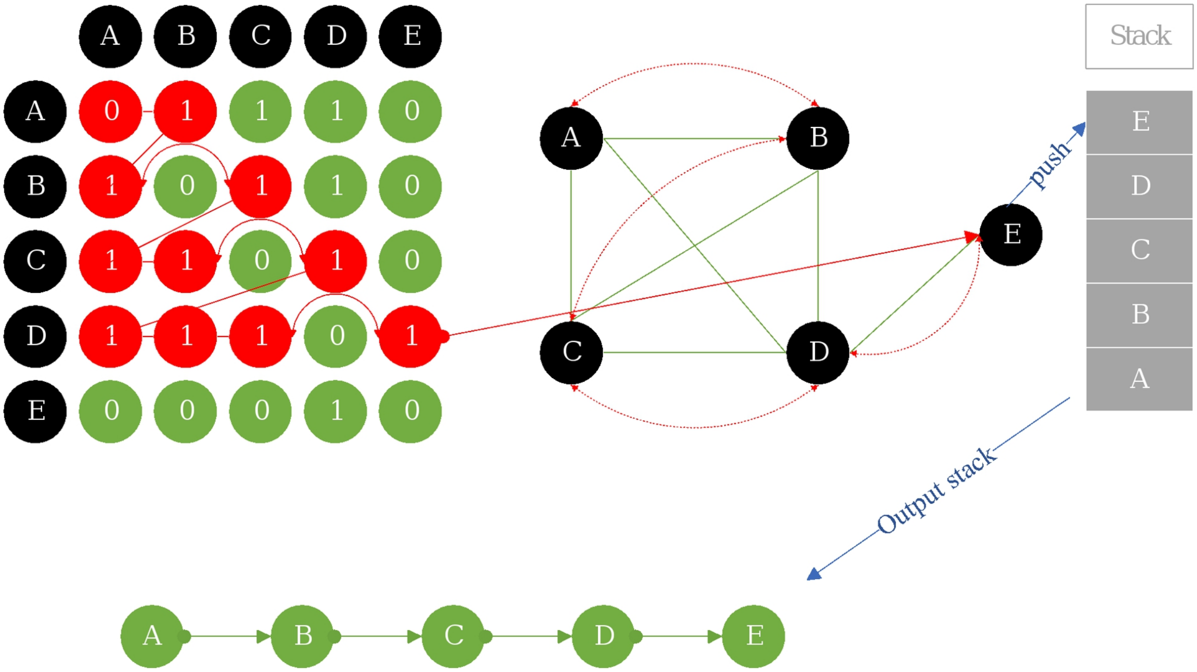
| | A | B | C | D | E |
|---|---|---|---|---|---|
| A | 0 | 1 | 1 | 1 | 0 |
| B | 1 | 0 | 1 | 1 | 0 |
| C | 1 | 1 | 0 | 1 | 0 |
| D | 1 | 1 | 1 | 0 | 1 |
| E | 0 | 0 | 0 | 1 | 0 |

A
B
C
D
E
push
Stack
C
B
A



| | A | B | C | D | E |
|---|---|---|---|---|---|
| A | 0 | 1 | 1 | 1 | 0 |
| B | 1 | 0 | 1 | 1 | 0 |
| C | 1 | 1 | 0 | 1 | 0 |
| D | 1 | 1 | 1 | 0 | 1 |
| E | 0 | 0 | 0 | 1 | 0 |

A
B
C
D
E
push
Stack
D
C
B
A



|  | A | B | C | D | E |
|---|---|---|---|---|---|
| A | 0 | 1 | 1 | 1 | 0 |
| B | 1 | 0 | 1 | 1 | 0 |
| C | 1 | 1 | 0 | 1 | 0 |
| D | 1 | 1 | 1 | 0 | 1 |
| E | 0 | 0 | 0 | 1 | 0 |

A
B
C
D
E
Stack
E
D
C
B
A
push
Output stack
A
B
C
D
E

**Create a TestUndirectedGraphDepthFrstSearch.html with Notepad**

```
<script type ="text/javascript" >
    class Vertex {
        constructor( data, visited){
            this.data = data;
            this.visited = visited; // Have you visited
        }

        getData() {
            return this.data;
        }
        setData( data) {
            this.data = data;
        }

        isVisited() {
            return this.visited;
        }
        setVisited( visited) {
            this.visited = visited;
        }
    }
```

```
}
```

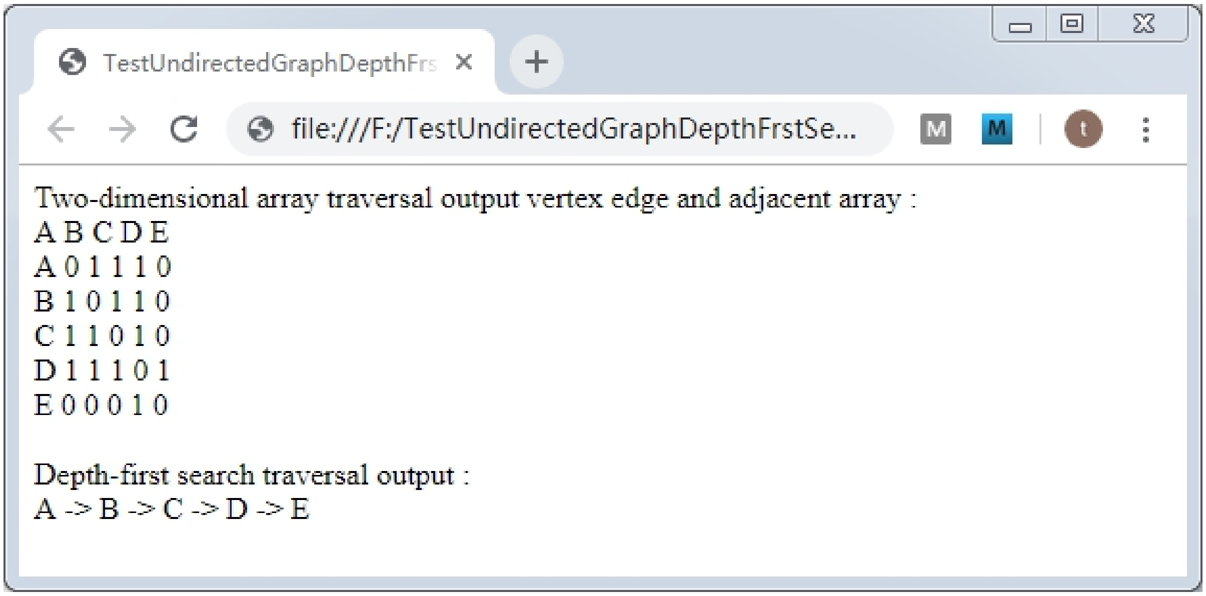
```
class Stack {
    constructor(){
        this.stacks = new Array();
        this.top = - 1 ;
    }

    push( element) {
        this.top++;
        this.stacks[ this.top] = element;
    }

    pop() {
        if ( this.top == - 1 ){
            return - 1 ;
        }

        var data = this.stacks[ this.top];
        this.top--;
        return data;
    }

    peek() {
        if ( this.top == - 1 ){
            return - 1 ;
        }

        var data = this.stacks[ this.top];
        return data;
    }
    isEmpty() {
        if ( this.top <= - 1 ){
            return true ;
        }
        return false ;
    }
}








class Graph {
    constructor( maxVertexSize){
        this.maxVertexSize = maxVertexSize; // Two-dimensional array size
        this.vertexs = new Array( maxVertexSize);
        this.size = 0 ; // Current vertex size
        this.adjacencyMatrix = new Array( maxVertexSize);
        for ( var i = 0 ; i < maxVertexSize; i++) {
            this.adjacencyMatrix[ i] = new Array();
            for ( var j = 0 ; j < maxVertexSize; j++) {
                this.adjacencyMatrix[ i][ j] = 0 ;
            }
        }
        this.stack = new Stack(); // Stack saves current vertices
    }

    addVertex( data) {
        var vertex = new Vertex( data, false );
        this.vertexs[ this.size] = vertex;
        this.size++;
    }

    // Add adjacent edges
```

```
addEdge( from, to) {
    // A -> B = B -> A
    this.adjacencyMatrix[ from][ to] = 1 ;
    this.adjacencyMatrix[ to][ from] = 1 ;
}
```

```
depthFirstSearch() {
    // Start searching from the first vertex
    var firstVertex = this.vertexs[ 0 ];
    firstVertex.setVisited( true );
    document.write( firstVertex.getData());
    this.stack.push( 0 );

    while (! this.stack.isEmpty()) {
        var row = this.stack.peek();
        // Get adjacent vertex positions that have not been visited
        var col = this.findAdjacencyUnVisitedVertex( row);
        if ( col == - 1 ) {
            this.stack.pop();
        } else {
```

```
                    this.vertexs[ col]. setVisited( true );
                    document.write( " -> " + this.vertexs[ col]. getData());
                    this.stack.push( col);
                }
            }
            this.clear();
        }

        // Get adjacent vertex positions that have not been visited
        findAdjacencyUnVisitedVertex( row) {
            for ( var col = 0 ; col < this.size; col++) {
                if ( this.adjacencyMatrix[ row][ col] == 1 && ! this.vertexs[ col].
isVisited()) {
                    return col;
                }
            }
            return - 1 ;
        }

        // Clear reset
        clear() {
            for ( var i = 0 ; i < this.size; i++) {
                this.vertexs[ i]. setVisited( false );
            }
        }

        getAdjacencyMatrix() {
            return this.adjacencyMatrix;
        }
        getVertexs() {
            return this.vertexs;
        }
    }

    //////////////////////testing////////////////////
    function printGraph( graph) {
        document.write( "Two-dimensional array traversal output vertex edge and adjacent array :  <br>" );
        for ( var i = 0 ; i < graph.getVertexs(). length; i++) {
            document.write( graph.getVertexs()[ i]. getData() + "  " );
        }
        document.write( "<br>" );

        for ( var i = 0 ; i < graph.getAdjacencyMatrix(). length; i++) {
            document.write( graph.getVertexs()[ i]. getData() + "  " );
            for ( var j = 0 ; j < graph.getAdjacencyMatrix(). length; j++) {
                document.write( graph.getAdjacencyMatrix()[ i][ j] + "  " );
            }
            document.write( "<br>" );
        }
    }

    var graph = new Graph( 5 );
    graph.addVertex( "A" );
    graph.addVertex( "B" );
    graph.addVertex( "C" );
    graph.addVertex( "D" );
    graph.addVertex( "E" );

    graph.addEdge( 0 , 1 );
    graph.addEdge( 0 , 2 );
    graph.addEdge( 0 , 3 );
    graph.addEdge( 1 , 2 );
    graph.addEdge( 1 , 3 );
    graph.addEdge( 2 , 3 );
    graph.addEdge( 3 , 4 );

    // Two-dimensional array traversal output vertex edge and adjacent array
    printGraph( graph);
    document.write( "<br>Depth-first search traversal output : <br>" );
    graph.depthFirstSearch();
</script>
```

**Result:**

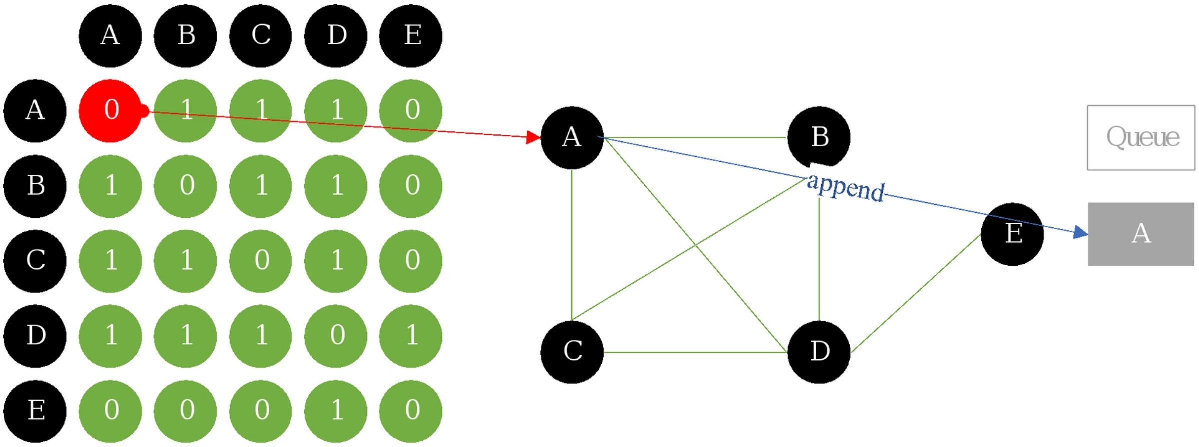

TestUndirectedGraphDepthFrs
file:///F:/TestUndirectedGraphDepthFrstSe...

```
Two-dimensional array traversal output vertex edge and adjacent array :
A B C D E
A 0 1 1 1 0
B 1 0 1 1 0
C 1 1 0 1 0
D 1 1 1 0 1
E 0 0 0 1 0

Depth-first search traversal output :
A -> B -> C -> D -> E
```

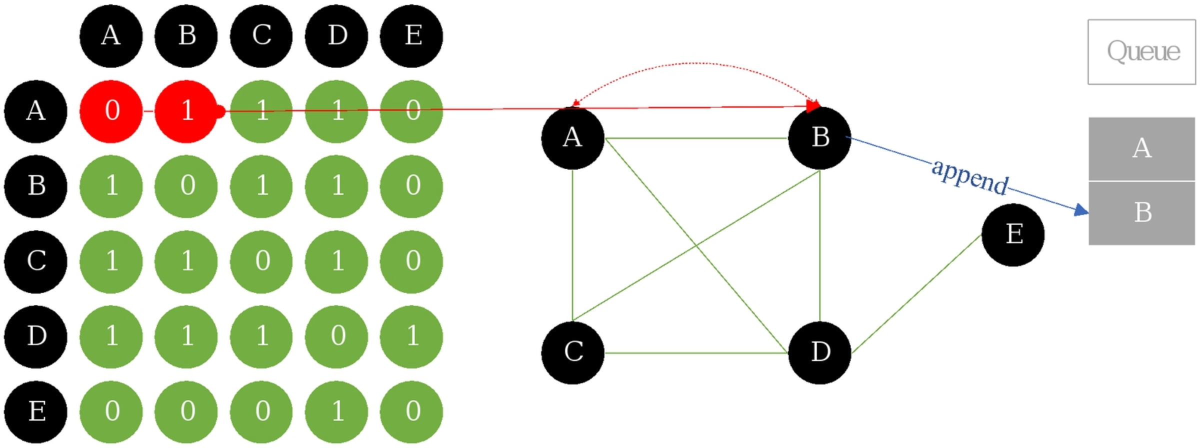
# Undirected Graph and Breadth-First Search

**Breadth-First Search:**

Find all neighboring edge nodes B, C, D from A and then find all neighboring nodes A, C, D from B and so on until all nodes are found A -> B -> C -> D -> E .

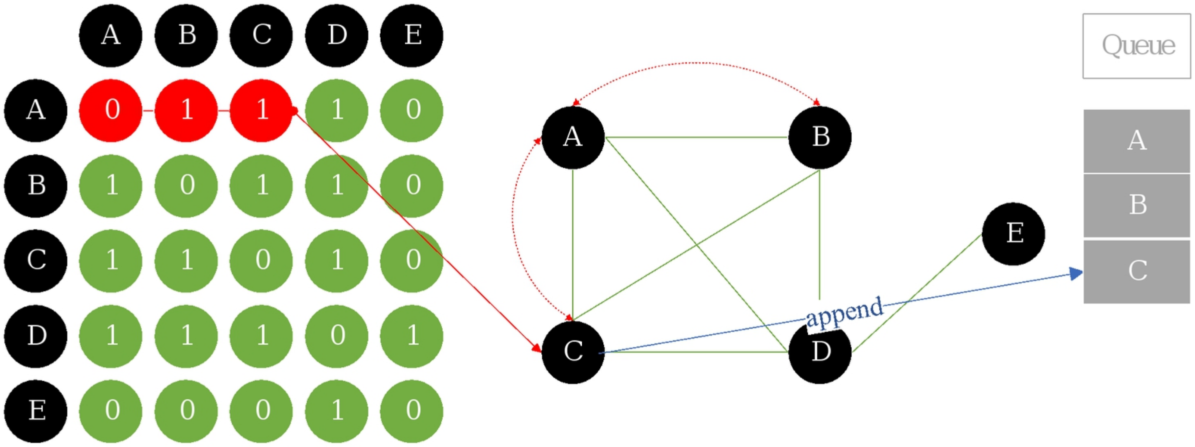


| | A | B | C | D | E |
|---|---|---|---|---|---|
| A | 0 | 1 | 1 | 1 | 0 |
| B | 1 | 0 | 1 | 1 | 0 |
| C | 1 | 1 | 0 | 1 | 0 |
| D | 1 | 1 | 1 | 0 | 1 |
| E | 0 | 0 | 0 | 1 | 0 |



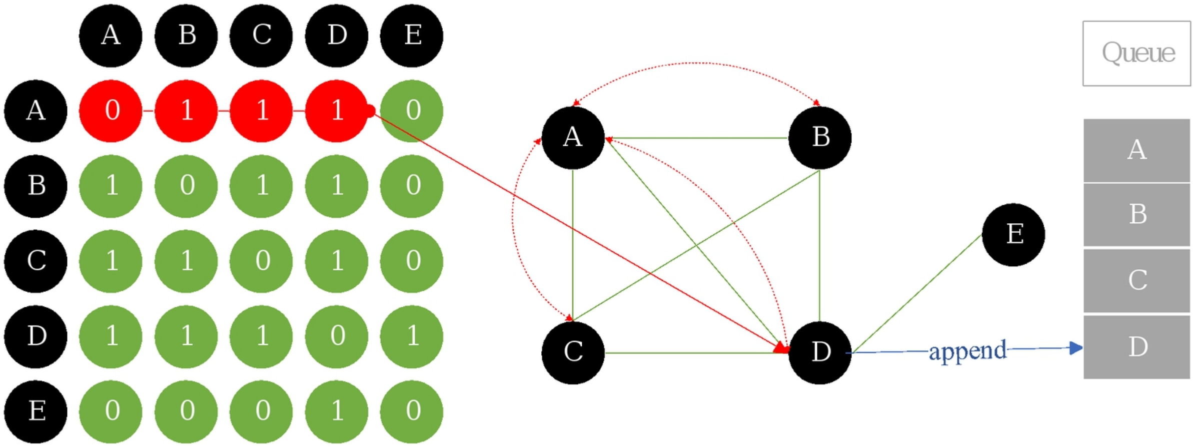
|   | A | B | C | D | E |
|---|---|---|---|---|---|
| A | 0 | 1 | 1 | 1 | 0 |
| B | 1 | 0 | 1 | 1 | 0 |
| C | 1 | 1 | 0 | 1 | 0 |
| D | 1 | 1 | 1 | 0 | 1 |
| E | 0 | 0 | 0 | 1 | 0 |

A
B
C
D
E
append
Queue
A
B



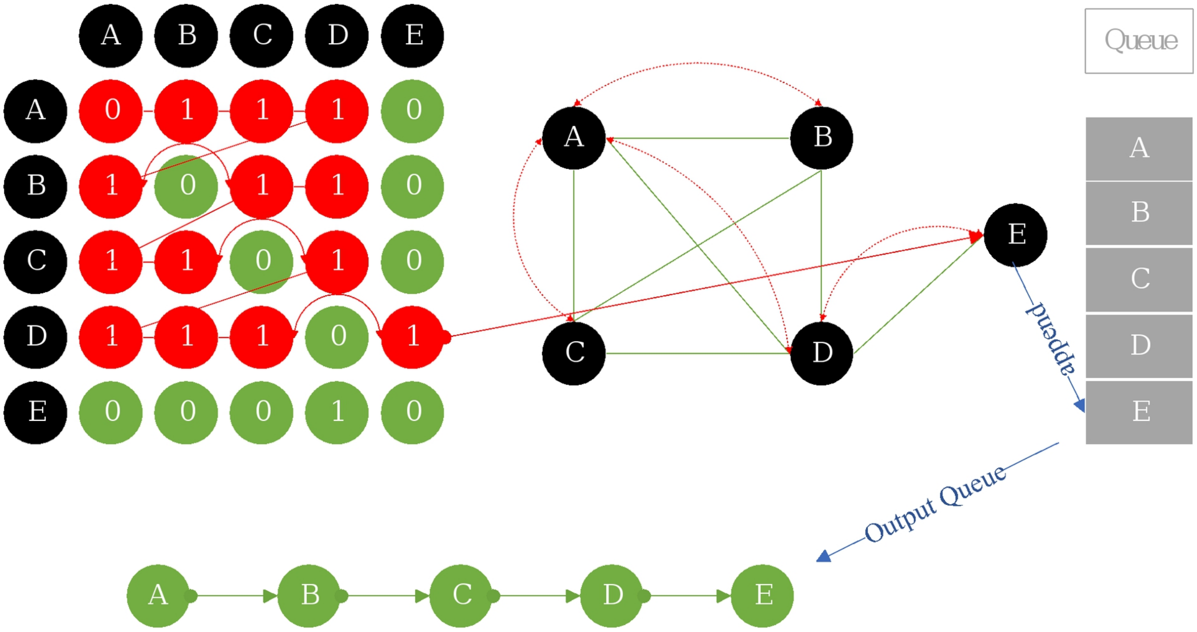
|   | A | B | C | D | E |
|---|---|---|---|---|---|
| A | 0 | 1 | 1 | 1 | 0 |
| B | 1 | 0 | 1 | 1 | 0 |
| C | 1 | 1 | 0 | 1 | 0 |
| D | 1 | 1 | 1 | 0 | 1 |
| E | 0 | 0 | 0 | 1 | 0 |

A
B
C
D
E
append
Queue
A
B
C



|   | A | B | C | D | E |
|---|---|---|---|---|---|
| A | 0 | 1 | 1 | 1 | 0 |
| B | 1 | 0 | 1 | 1 | 0 |
| C | 1 | 1 | 0 | 1 | 0 |
| D | 1 | 1 | 1 | 0 | 1 |
| E | 0 | 0 | 0 | 1 | 0 |

A
B
C
D
E
append
Queue
A
B
C
D



| | A | B | C | D | E |
|---|---|---|---|---|---|
| A | 0 | 1 | 1 | 1 | 0 |
| B | 1 | 0 | 1 | 1 | 0 |
| C | 1 | 1 | 0 | 1 | 0 |
| D | 1 | 1 | 1 | 0 | 1 |
| E | 0 | 0 | 0 | 1 | 0 |

A
B
C
D
E
Queue
A
B
C
D
E
append
Output Queue
A
B
C
D
E

**Create a TestUndirectedGraphBreadthFrstSearch.html with Notepad**

```
<script type ="text/javascript" >
  class Vertex {
    constructor( data, visited){
      this.data = data;
      this.visited = visited; // Have you visited
    }

    getData() {
      return this.data;
    }
    setData( data) {
      this.data = data;
    }

    isVisited() {
      return this.visited;
    }
    setVisited( visited) {
      this.visited = visited;
    }
  }
```

```
class Queue {
    constructor(){
        this.queues = new Array();
    }

    add( element) {
        this.queues[ this.queues.length] = element;
    }

    remove() {
        if ( this.queues.length <= 0 ){
            return - 1 ;
        }
        return this.queues.shift();
    }

    isEmpty() {
        if ( this.queues.length <= 0 ){
            return true ;
        }
        return false ;
    }
}
```

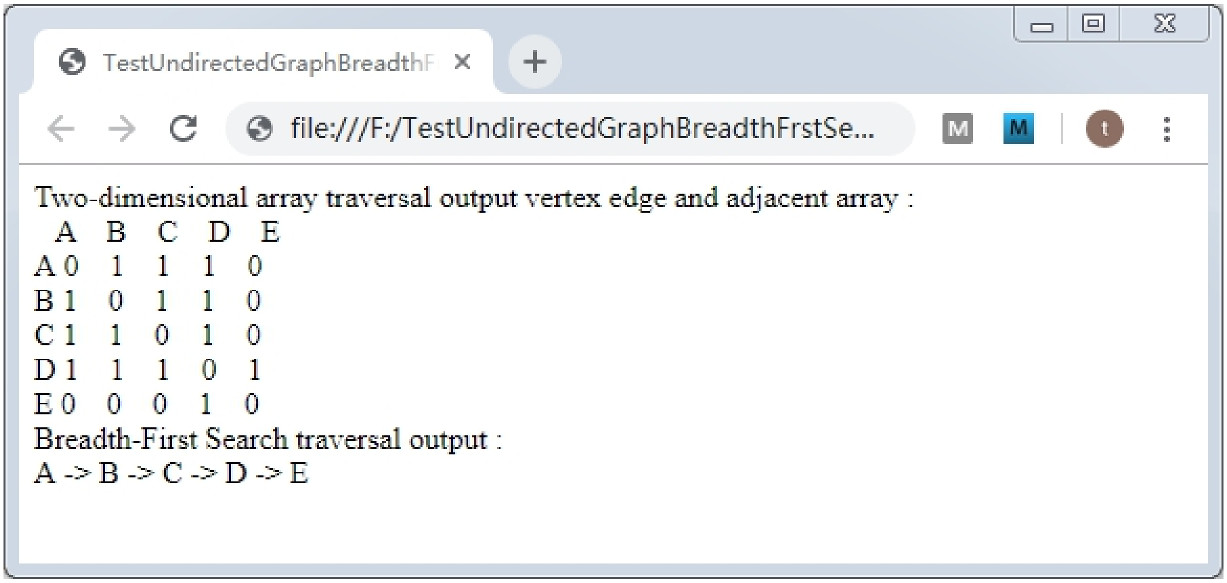
```
class Graph {
    constructor( maxVertexSize){
        this.maxVertexSize = maxVertexSize; // Two-dimensional array size
        this.vertexs = new Array( maxVertexSize);
        this.size = 0 ; // Current vertex size
        this.adjacencyMatrix = new Array( maxVertexSize);
        for ( var i = 0 ; i < maxVertexSize; i++) {
            this.adjacencyMatrix[ i] = new Array();
            for ( var j = 0 ; j < maxVertexSize; j++) {
                this.adjacencyMatrix[ i][ j] = 0 ;
            }
        }
        this.queue = new Queue(); // Queue  saves current vertices
    }

    addVertex( data) {
        var vertex = new Vertex( data, false );
        this.vertexs[ this.size] = vertex;
        this.size++;
    }
```

```
// Add adjacent edges
addEdge( from, to) {
    // A -> B = B -> A
    this.adjacencyMatrix[ from][ to] = 1 ;
    this.adjacencyMatrix[ to][ from] = 1 ;
}



breadthFirstSearch() {
    // Start searching from the first vertex
    var firstVertex = this.vertexs[ 0 ];
    firstVertex.setVisited( true );
    document.write( firstVertex.getData());
    this.queue.add( 0 );

    var col = 0 ;
    while (! this.queue.isEmpty()) {
        var head = this.queue.remove();
     // Get adjacent vertex positions that have not been visited
        col = this.findAdjacencyUnVisitedVertex( head);
```

```
       //Loop through all vertices connected to the current vertex
        while ( col != - 1 )
        {
          this.vertexs[ col]. setVisited( true );
          document.write( " -> " + this.vertexs[ col]. getData());
          this.queue.add( col);
          col = this.findAdjacencyUnVisitedVertex( head);
        }
      }
      this.clear();
    }

    // Get adjacent vertex positions that have not been visited
    findAdjacencyUnVisitedVertex( row) {
      for ( var col = 0 ; col < this.size; col++) {
        if ( this.adjacencyMatrix[ row][ col] == 1 && ! this.vertexs[ col].
isVisited()) {
          return col;
        }
      }
      return - 1 ;
    }

    // Clear reset
    clear() {
      for ( var i = 0 ; i < this.size; i++) {
        this.vertexs[ i]. setVisited( false );
      }
    }



    getAdjacencyMatrix() {
      return this.adjacencyMatrix;
    }
```

```
        getVertexs() {
            return this.vertexs;
        }
    }

    ////////////////////testing//////////////////
    function printGraph( graph) {
        document.write( "Two-dimensional array traversal output vertex edge
and adjacent array :  <br>" );
        document.write( "&nbsp;&nbsp;&nbsp;" );
        for ( var i = 0 ; i < graph.getVertexs(). length; i++) {
            document.write( graph.getVertexs()[ i]. getData() +
"&nbsp;&nbsp;&nbsp;&nbsp;" );
        }
        document.write( "<br>" );

        for ( var i = 0 ; i < graph.getAdjacencyMatrix(). length; i++) {
            document.write( graph.getVertexs()[ i]. getData() + "  " );
            for ( var j = 0 ; j < graph.getAdjacencyMatrix(). length; j++) {
                document.write( graph.getAdjacencyMatrix()[ i][ j] +
"&nbsp;&nbsp;&nbsp;&nbsp;" );
            }
            document.write( "<br>" );
        }
    }

    var graph = new Graph( 5 );
    graph.addVertex( "A" );
    graph.addVertex( "B" );
    graph.addVertex( "C" );
    graph.addVertex( "D" );
    graph.addVertex( "E" );

    graph.addEdge( 0 , 1 );
    graph.addEdge( 0 , 2 );
    graph.addEdge( 0 , 3 );
    graph.addEdge( 1 , 2 );
    graph.addEdge( 1 , 3 );
    graph.addEdge( 2 , 3 );
    graph.addEdge( 3 , 4 );

    // Two-dimensional array traversal output vertex edge and adjacent array
    printGraph( graph);

    document.write( "\nBreadth-First Search traversal output : <br>" );
    graph.breadthFirstSearch();

</script>
```

**Result:**

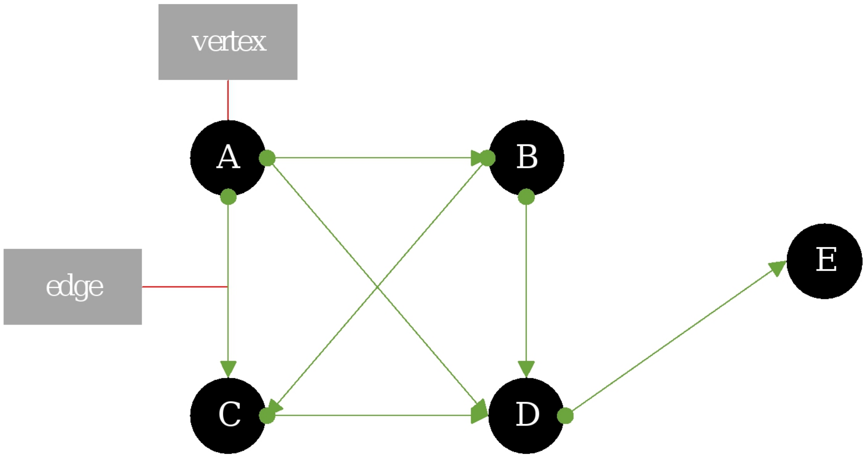

TestUndirectedGraphBreadthF
file:///F:/TestUndirectedGraphBreadthFrstSe...

```
Two-dimensional array traversal output vertex edge and adjacent array :
  A  B  C  D  E
A 0  1  1  1  0
B 1  0  1  1  0
C 1  1  0  1  0
D 1  1  1  0  1
E 0  0  0  1  0
Breadth-First Search traversal output :
A -> B -> C -> D -> E
```

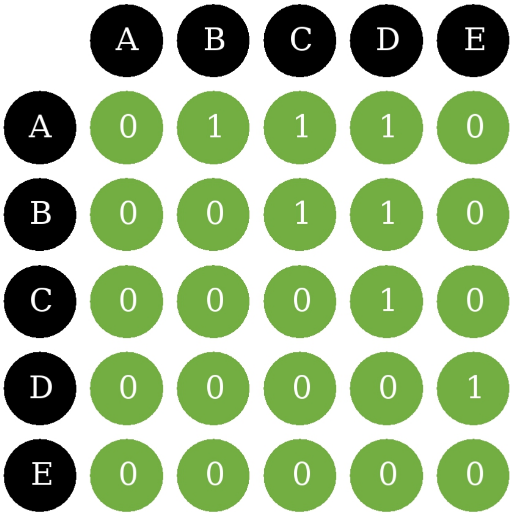
# Directed Graph and Depth-Frst Search

**Directed Graph:**
The data structure is represented by an adjacency matrix (that is, a two-dimensional array) and an adjacency list. Each node is called a vertex, and two adjacent nodes are called edges.

**Directed Graph** has direction : A -> B and B -> A are different



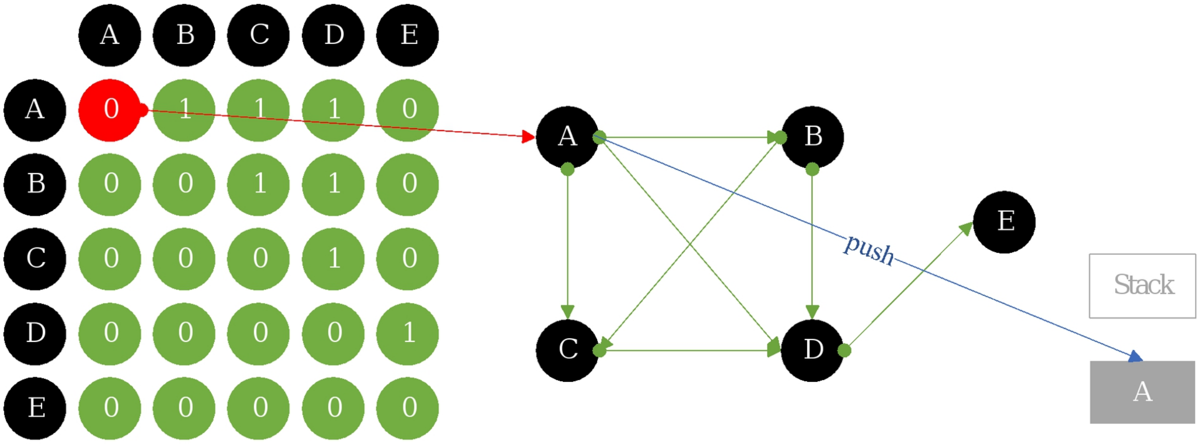
**1. The adjacency matrix is described above:**
The total number of vertices is a two-dimensional array size, if have value of the edge is 1 , otherwise no value of the edge is 0 .



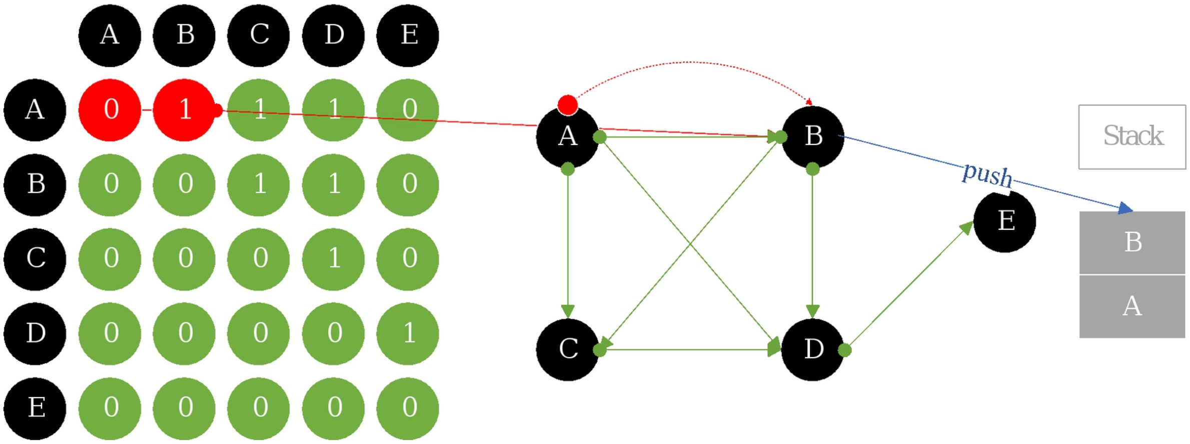
|   | A | B | C | D | E |
|---|---|---|---|---|---|
| A | 0 | 1 | 1 | 1 | 0 |
| B | 0 | 0 | 1 | 1 | 0 |
| C | 0 | 0 | 0 | 1 | 0 |
| D | 0 | 0 | 0 | 0 | 1 |
| E | 0 | 0 | 0 | 0 | 0 |

## 2. Depth-Frst Search:

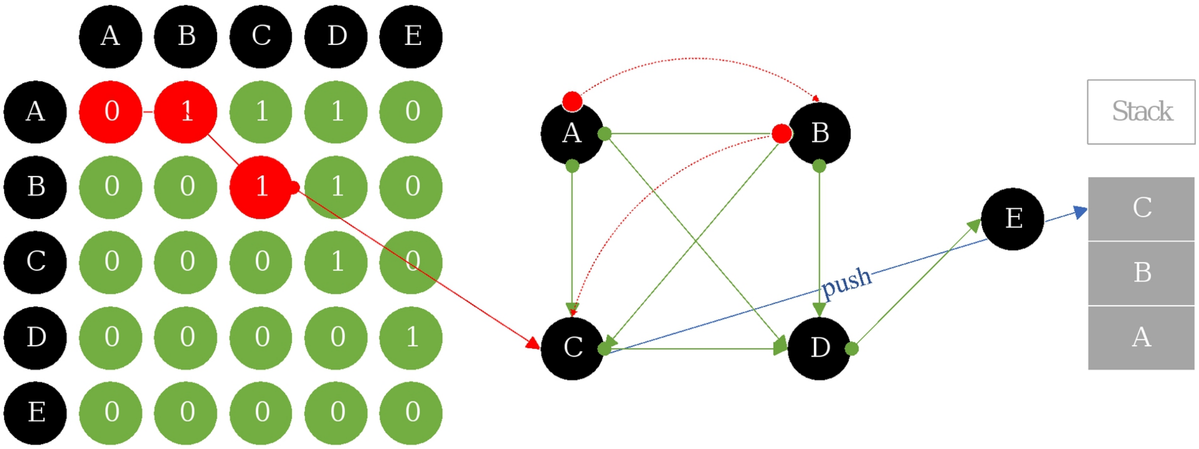
Look for the neighboring edge node B from A and then find the neighboring node C from B and so on until all nodes are found A -> B -> C -> D -> E .


A
B
C
D
E
push
Stack
A

|   | A | B | C | D | E |
|---|---|---|---|---|---|
| A | 0 | 1 | 1 | 1 | 0 |
| B | 0 | 0 | 1 | 1 | 0 |
| C | 0 | 0 | 0 | 1 | 0 |
| D | 0 | 0 | 0 | 0 | 1 |
| E | 0 | 0 | 0 | 0 | 0 |

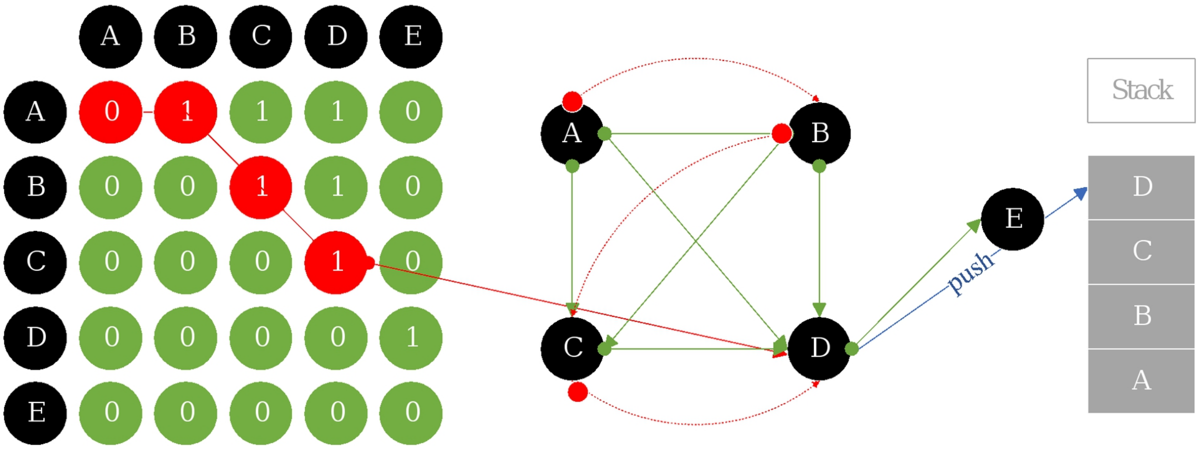

A
B
C
D
E
push
Stack
B
A

|   | A | B | C | D | E |
|---|---|---|---|---|---|
| A | 0 | 1 | 1 | 1 | 0 |
| B | 0 | 0 | 1 | 1 | 0 |
| C | 0 | 0 | 0 | 1 | 0 |
| D | 0 | 0 | 0 | 0 | 1 |
| E | 0 | 0 | 0 | 0 | 0 |



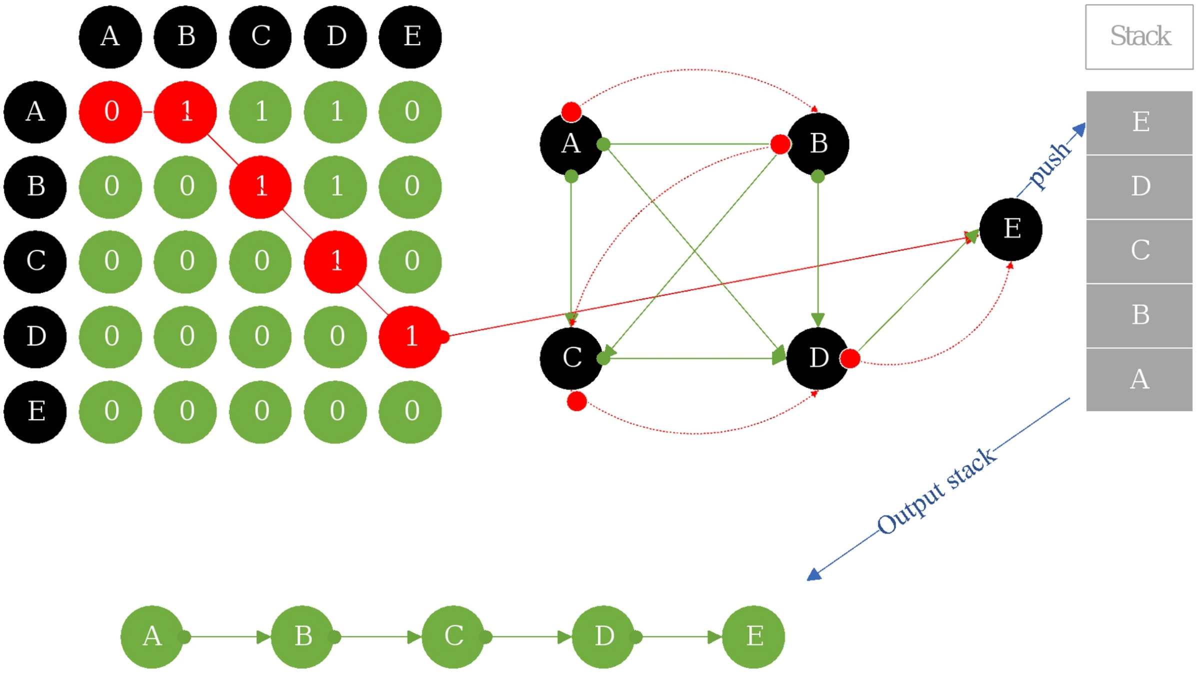
| | A | B | C | D | E |
|---|---|---|---|---|---|
| A | 0 | 1 | 1 | 1 | 0 |
| B | 0 | 0 | 1 | 1 | 0 |
| C | 0 | 0 | 0 | 1 | 0 |
| D | 0 | 0 | 0 | 0 | 1 |
| E | 0 | 0 | 0 | 0 | 0 |

A
B
C
D
E
push
Stack
C
B
A



| | A | B | C | D | E |
|---|---|---|---|---|---|
| A | 0 | 1 | 1 | 1 | 0 |
| B | 0 | 0 | 1 | 1 | 0 |
| C | 0 | 0 | 0 | 1 | 0 |
| D | 0 | 0 | 0 | 0 | 1 |
| E | 0 | 0 | 0 | 0 | 0 |

A
B
C
D
E
push
Stack
D
C
B
A



| | A | B | C | D | E |
|---|---|---|---|---|---|
| A | 0 | 1 | 1 | 1 | 0 |
| B | 0 | 0 | 1 | 1 | 0 |
| C | 0 | 0 | 0 | 1 | 0 |
| D | 0 | 0 | 0 | 0 | 1 |
| E | 0 | 0 | 0 | 0 | 0 |

A
B
C
D
E
push
Stack
E
D
C
B
A
Output stack
A
B
C
D
E

**Create a TestDirectedGraphDepthFrstSearch.html with Notepad**

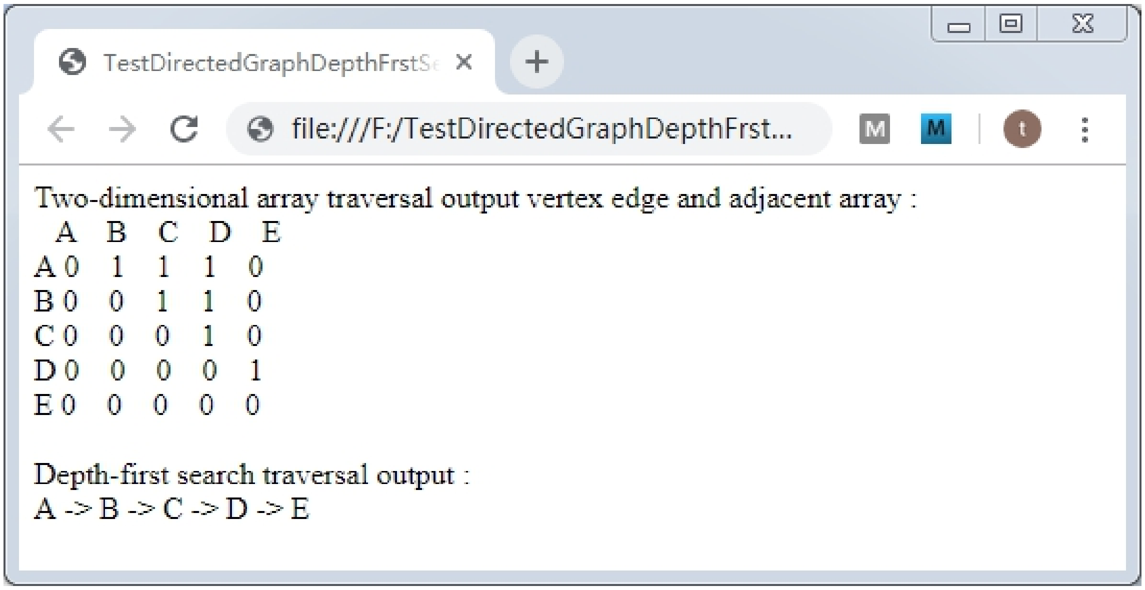
```
<script type ="text/javascript" >
    class Vertex {
        constructor( data, visited){
            this.data = data;
            this.visited = visited; // Have you visited
        }

        getData() {
            return this.data;
        }
        setData( data) {
            this.data = data;
        }

        isVisited() {
            return this.visited;
        }
        setVisited( visited) {
            this.visited = visited;
        }
    }
    }
```

```
class Stack {
    constructor(){
        this.stacks = new Array();
        this.top = - 1 ;
    }

    push( element) {
        this.top++;
        this.stacks[ this.top] = element;
    }

    pop() {
        if ( this.top == - 1 ){
            return - 1 ;
        }

        var data = this.stacks[ this.top];
        this.top--;
        return data;
    }

    peek() {
        if ( this.top == - 1 ){
            return - 1 ;
        }

        var data = this.stacks[ this.top];
        return data;
    }
    isEmpty() {
        if ( this.top <= - 1 ){
            return true ;
        }
        return false ;
    }
}







class Graph {
    constructor( maxVertexSize){
        this.maxVertexSize = maxVertexSize; // Two-dimensional array size
        this.vertexs = new Array( maxVertexSize);
        this.size = 0 ; // Current vertex size
        this.adjacencyMatrix = new Array( maxVertexSize);
        for ( var i = 0 ; i < maxVertexSize; i++) {
            this.adjacencyMatrix[ i] = new Array();
            for ( var j = 0 ; j < maxVertexSize; j++) {
                this.adjacencyMatrix[ i][ j] = 0 ;
            }
        }
        this.stack = new Stack(); // Stack saves current vertices
    }

    addVertex( data) {
        var vertex = new Vertex( data, false );
        this.vertexs[ this.size++] = vertex;
    }

    addEdge( from, to) {
        this.adjacencyMatrix[ from][ to] = 1 ; // A -> B  and B -> A are
different
```

```
    }

    depthFirstSearch() {
        var firstVertex = this.vertexs[ 0 ]; // Start searching from the first vertex
        firstVertex.setVisited( true );
        document.write( firstVertex.getData());
        this.stack.push( 0 );
        while (! this.stack.isEmpty()) {
            var row = this.stack.peek();
            // Get adjacent vertex positions that have not been visited
            var col = this.findAdjacencyUnVisitedVertex( row);
            if ( col == - 1 ) {
                this.stack.pop();
            } else {
                this.vertexs[ col]. setVisited( true );
                document.write( " -> " + this.vertexs[ col]. getData());
                this.stack.push( col);
            }
        }
        this.clear();
    }

    // Get adjacent vertex positions that have not been visited
    findAdjacencyUnVisitedVertex( row) {
        for ( var col = 0 ; col < this.size; col++) {
            if ( this.adjacencyMatrix[ row][ col] == 1 && ! this.vertexs[ col]. isVisited()) {
                return col;
            }
        }

        return - 1 ;
    }

    // Clear reset
    clear() {
          for ( var i = 0 ; i < this.size; i++) {
              this.vertexs[ i]. setVisited( false );
          }
      }

      getAdjacencyMatrix() {
          return this.adjacencyMatrix;
      }

      getVertexs() {
          return this.vertexs;
      }
  }

  //////////////////////testing///////////////////
  function printGraph( graph) {
      document.write( "Two-dimensional array traversal output vertex edge
and adjacent array :  <br>" );
      document.write( "&nbsp;&nbsp;&nbsp;" );
```

```
        for ( var i = 0 ; i < graph.getVertexs(). length; i++) {
            document.write( graph.getVertexs()[ i]. getData() +
"&nbsp;&nbsp;&nbsp;&nbsp;" );
        }
        document.write( "<br>" );

        for ( var i = 0 ; i < graph.getAdjacencyMatrix(). length; i++) {
            document.write( graph.getVertexs()[ i]. getData() + "  " );
            for ( var j = 0 ; j < graph.getAdjacencyMatrix(). length; j++) {
                document.write( graph.getAdjacencyMatrix()[ i][ j] +
"&nbsp;&nbsp;&nbsp;&nbsp;" );
            }
            document.write( "<br>" );
        }
    }

    var graph = new Graph( 5 );
    graph.addVertex( "A" );
    graph.addVertex( "B" );
    graph.addVertex( "C" );
    graph.addVertex( "D" );
    graph.addVertex( "E" );

    graph.addEdge( 0 , 1 );
    graph.addEdge( 0 , 2 );
    graph.addEdge( 0 , 3 );
    graph.addEdge( 1 , 2 );
    graph.addEdge( 1 , 3 );
    graph.addEdge( 2 , 3 );
    graph.addEdge( 3 , 4 );

    // Two-dimensional array traversal output vertex edge and adjacent array
    printGraph( graph);
    document.write( "<br>Depth-first search traversal output : <br>" );
    graph.depthFirstSearch();
</script>
```

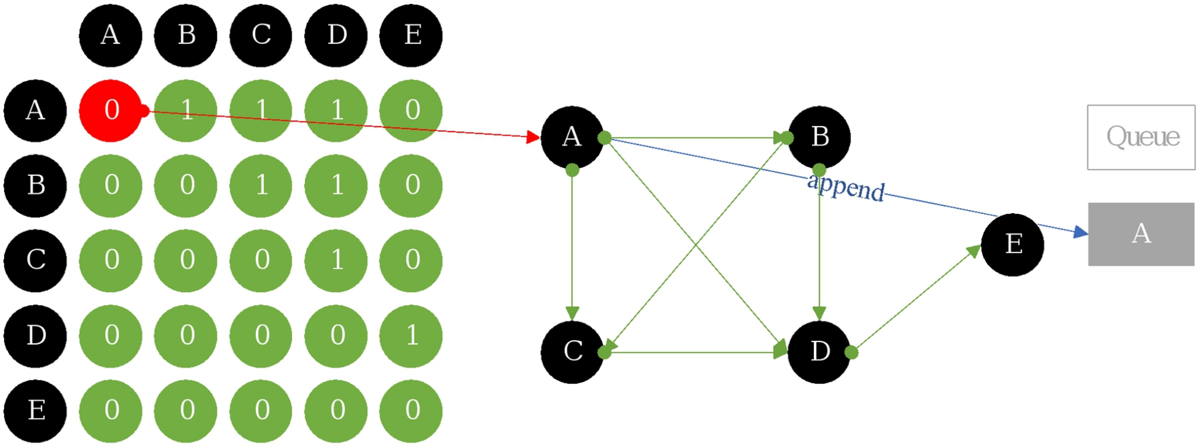
**Result:**


TestDirectedGraphDepthFrstSe
file:///F:/TestDirectedGraphDepthFrst...

```
Two-dimensional array traversal output vertex edge and adjacent array :
   A   B   C   D   E
A 0   1   1   1   0
B 0   0   1   1   0
C 0   0   0   1   0
D 0   0   0   0   1
E 0   0   0   0   0

Depth-first search traversal output :
A -> B -> C -> D -> E
```

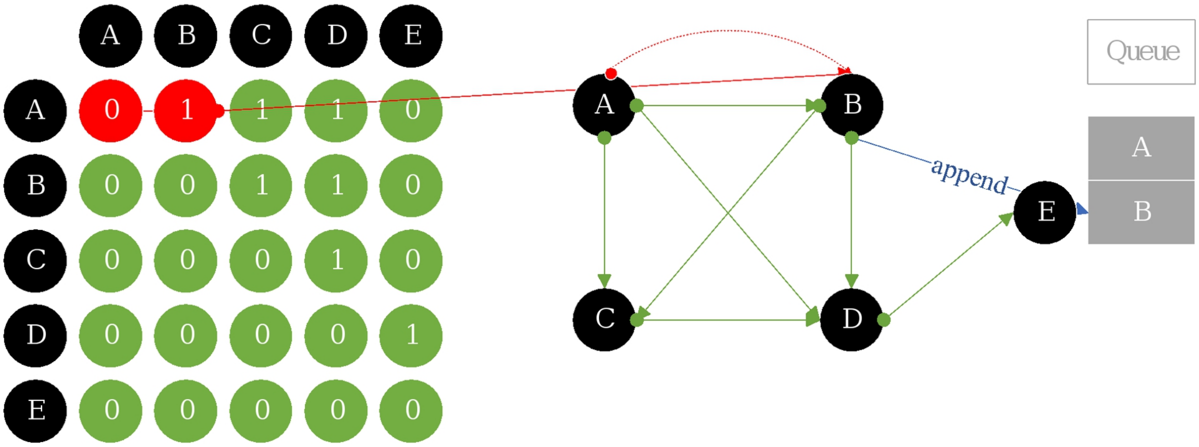
# Directed Graph and Breadth-First Search

**Breadth-First Search:**
Find all neighboring edge nodes B, C, D from A and then find all neighboring nodes A, C, D from B and so on until all nodes are found A -> B -> C -> D -> E .



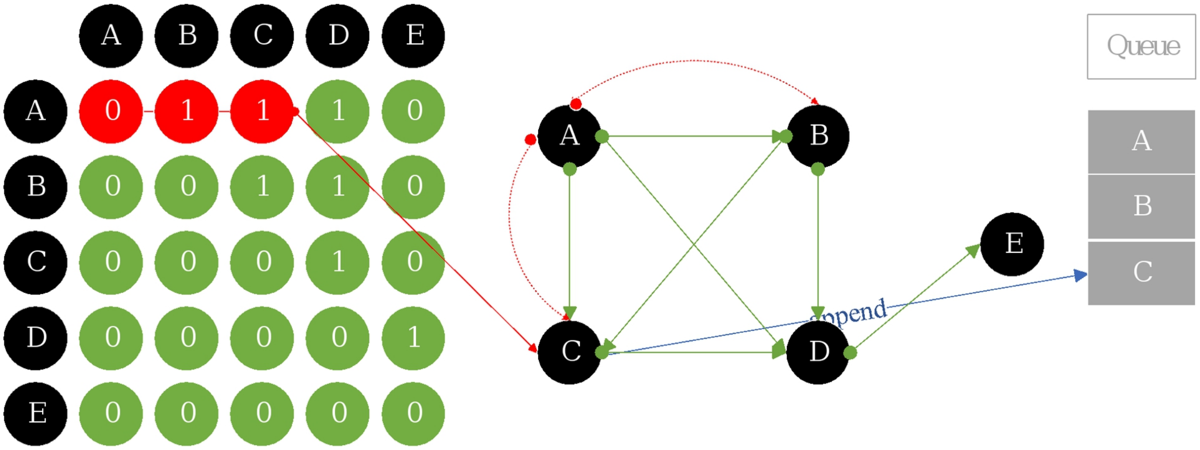
|   | A | B | C | D | E |
|---|---|---|---|---|---|
| A | 0 | 1 | 1 | 1 | 0 |
| B | 0 | 0 | 1 | 1 | 0 |
| C | 0 | 0 | 0 | 1 | 0 |
| D | 0 | 0 | 0 | 0 | 1 |
| E | 0 | 0 | 0 | 0 | 0 |

A
B
C
D
E
append
Queue
A

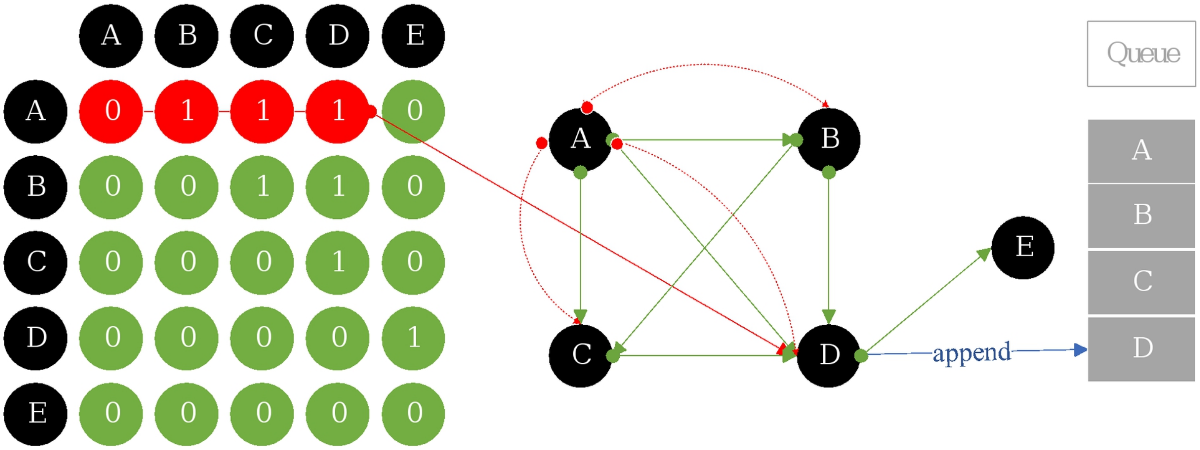


|   | A | B | C | D | E |
|---|---|---|---|---|---|
| A | 0 | 1 | 1 | 1 | 0 |
| B | 0 | 0 | 1 | 1 | 0 |
| C | 0 | 0 | 0 | 1 | 0 |
| D | 0 | 0 | 0 | 0 | 1 |
| E | 0 | 0 | 0 | 0 | 0 |

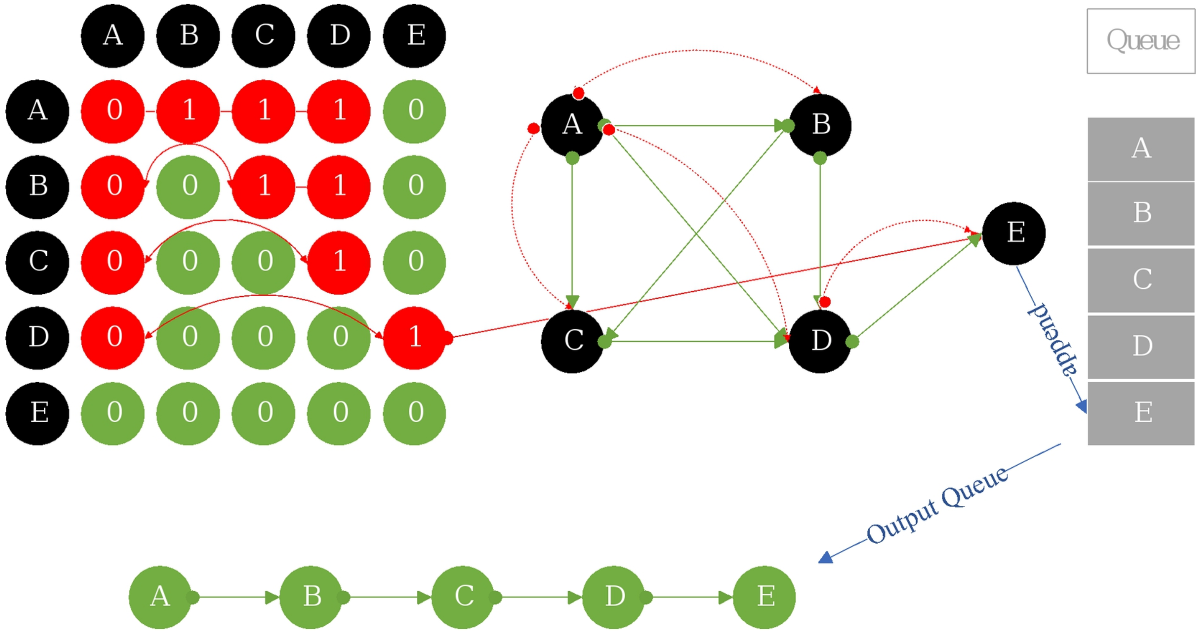
A
B
C
D
E
append
Queue
A
B


A
B
C
D
E
append
Queue
A
B
C

|  | A | B | C | D | E |
|---|---|---|---|---|---|
| A | 0 | 1 | 1 | 1 | 0 |
| B | 0 | 0 | 1 | 1 | 0 |
| C | 0 | 0 | 0 | 1 | 0 |
| D | 0 | 0 | 0 | 0 | 1 |
| E | 0 | 0 | 0 | 0 | 0 |


A
B
C
D
E
append
Queue
A
B
C
D

|  | A | B | C | D | E |
|---|---|---|---|---|---|
| A | 0 | 1 | 1 | 1 | 0 |
| B | 0 | 0 | 1 | 1 | 0 |
| C | 0 | 0 | 0 | 1 | 0 |
| D | 0 | 0 | 0 | 0 | 1 |
| E | 0 | 0 | 0 | 0 | 0 |


A
B
C
D
E
append
Queue
A
B
C
D
E
Output Queue
A
B
C
D
E

|  | A | B | C | D | E |
|---|---|---|---|---|---|
| A | 0 | 1 | 1 | 1 | 0 |
| B | 0 | 0 | 1 | 1 | 0 |
| C | 0 | 0 | 0 | 1 | 0 |
| D | 0 | 0 | 0 | 0 | 1 |
| E | 0 | 0 | 0 | 0 | 0 |

**Create a TestDirectedGraphBreadthFrstSearch.html with Notepad**

```
<script type ="text/javascript" >
    class Vertex {
        constructor( data, visited){
            this.data = data;
            this.visited = visited; // Have you visited
        }

        getData() {
            return this.data;
        }
        setData( data) {
            this.data = data;
        }

        isVisited() {
            return this.visited;
        }
        setVisited( visited) {
            this.visited = visited;
        }
    }
```

```
class Queue {
    constructor(){
        this.queues = new Array();
    }

    add( element) {
        this.queues[ this.queues.length] = element;
    }

    remove() {
        if ( this.queues.length <= 0 ){
            return - 1 ;
        }
        return this.queues.shift();
    }

    isEmpty() {
        if ( this.queues.length <= 0 ){
            return true ;
        }
        return false ;
    }
}
```

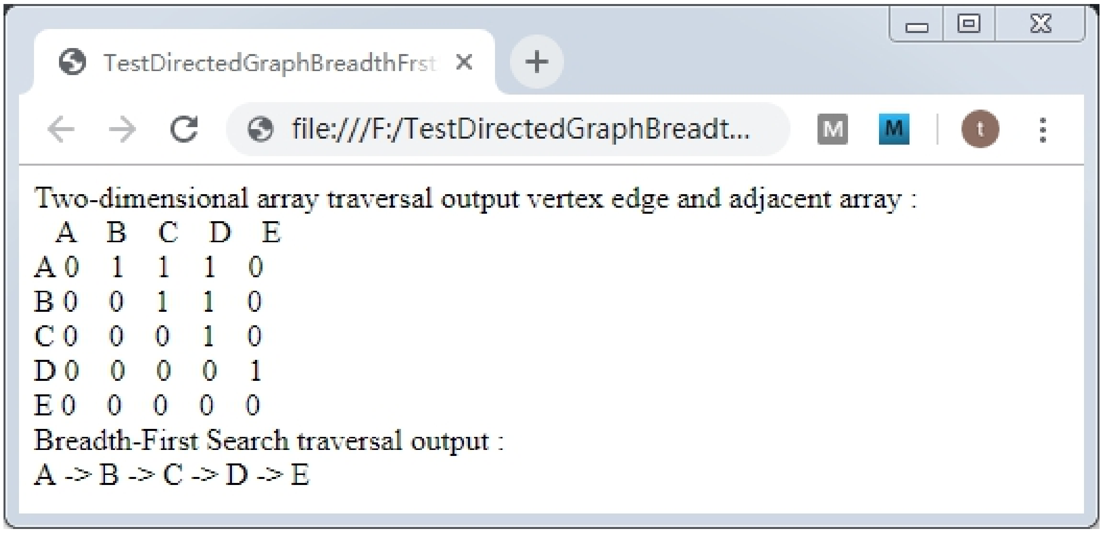
```
class Graph {
    constructor( maxVertexSize){
        this.maxVertexSize = maxVertexSize; // Two-dimensional array size
        this.vertexs = new Array( maxVertexSize);
        this.size = 0 ; // Current vertex size
        this.adjacencyMatrix = new Array( maxVertexSize);
        for ( var i = 0 ; i < maxVertexSize; i++) {
            this.adjacencyMatrix[ i] = new Array();
            for ( var j = 0 ; j < maxVertexSize; j++) {
                this.adjacencyMatrix[ i][ j] = 0 ;
            }
        }
        this.queue = new Queue(); // Queue  saves current vertices
    }

    addVertex( data) {
        var vertex = new Vertex( data, false );
        this.vertexs[ this.size] = vertex;
        this.size++;
```

```
}

// Add adjacent edges
addEdge( from, to) {
    // A -> B  and B -> A are different
    this.adjacencyMatrix[ from][ to] = 1 ;
}
















breadthFirstSearch() {
    // Start searching from the first vertex
    var firstVertex = this.vertexs[ 0 ];
    firstVertex.setVisited( true );
    document.write( firstVertex.getData());
    this.queue.add( 0 );

    var col = 0 ;
    while (! this.queue.isEmpty()) {
        var head = this.queue.remove();
     // Get adjacent vertex positions that have not been visited
```

```
        col = this.findAdjacencyUnVisitedVertex( head);

       //Loop through all vertices connected to the current vertex
        while ( col != - 1 )
        {
          this.vertexs[ col]. setVisited( true );
          document.write( " -> " + this.vertexs[ col]. getData());
          this.queue.add( col);
          col = this.findAdjacencyUnVisitedVertex( head);
        }
      }
      this.clear();
    }

    // Get adjacent vertex positions that have not been visited
    findAdjacencyUnVisitedVertex( row) {
      for ( var col = 0 ; col < this.size; col++) {
        if ( this.adjacencyMatrix[ row][ col] == 1 && ! this.vertexs[ col].
isVisited()) {
          return col;
        }
      }
      return - 1 ;
    }

    // Clear reset
    clear() {
      for ( var i = 0 ; i < this.size; i++) {
        this.vertexs[ i]. setVisited( false );
      }
    }


    getAdjacencyMatrix() {
      return this.adjacencyMatrix;
    }
```

```
        getVertexs() {
            return this.vertexs;
        }
    }

    ////////////////////testing//////////////////
    function printGraph( graph) {
        document.write( "Two-dimensional array traversal output vertex edge
and adjacent array :  <br>" );
        document.write( "&nbsp;&nbsp;&nbsp;" );
        for ( var i = 0 ; i < graph.getVertexs(). length; i++) {
            document.write( graph.getVertexs()[ i]. getData() +
"&nbsp;&nbsp;&nbsp;&nbsp;" );
        }
        document.write( "<br>" );

        for ( var i = 0 ; i < graph.getAdjacencyMatrix(). length; i++) {
            document.write( graph.getVertexs()[ i]. getData() + "  " );
            for ( var j = 0 ; j < graph.getAdjacencyMatrix(). length; j++) {
                document.write( graph.getAdjacencyMatrix()[ i][ j] +
"&nbsp;&nbsp;&nbsp;&nbsp;" );
            }
            document.write( "<br>" );
        }
    }

    var graph = new Graph( 5 );
    graph.addVertex( "A" );
    graph.addVertex( "B" );
    graph.addVertex( "C" );
    graph.addVertex( "D" );
    graph.addVertex( "E" );

    graph.addEdge( 0 , 1 );
    graph.addEdge( 0 , 2 );
    graph.addEdge( 0 , 3 );
    graph.addEdge( 1 , 2 );
    graph.addEdge( 1 , 3 );
    graph.addEdge( 2 , 3 );
    graph.addEdge( 3 , 4 );

    // Two-dimensional array traversal output vertex edge and adjacent array
    printGraph( graph);

    document.write( "\nBreadth-First Search traversal output : <br>" );
    graph.breadthFirstSearch();

</script>
```

**Result:**

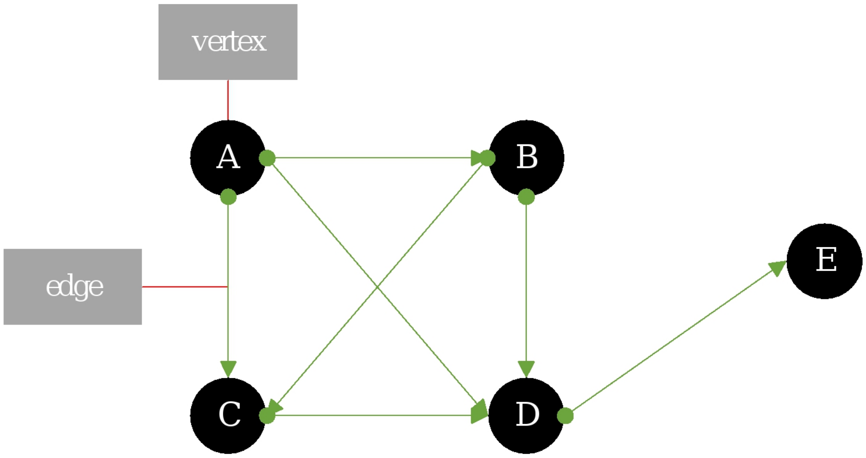

TestDirectedGraphBreadthFrst
file:///F:/TestDirectedGraphBreadt...

```
Two-dimensional array traversal output vertex edge and adjacent array :
  A  B  C  D  E
A 0  1  1  1  0
B 0  0  1  1  0
C 0  0  0  1  0
D 0  0  0  0  1
E 0  0  0  0  0
Breadth-First Search traversal output :
A -> B -> C -> D -> E
```

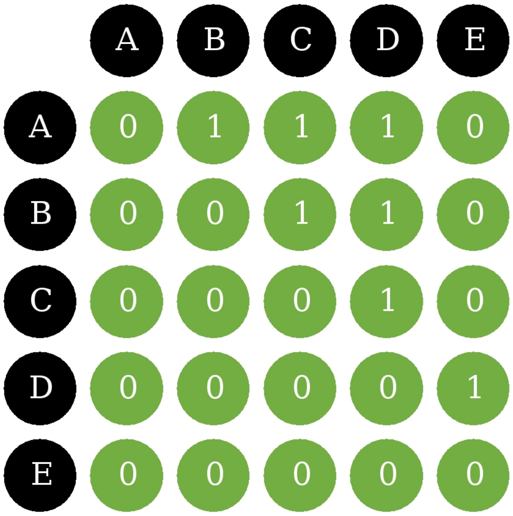
# Directed Graph Topological Sorting

**Directed Graph Topological Sorting:**
Sort the vertices in the directed graph with order of direction
.

**Directed Graph** has direction : A -> B and B -> A are different



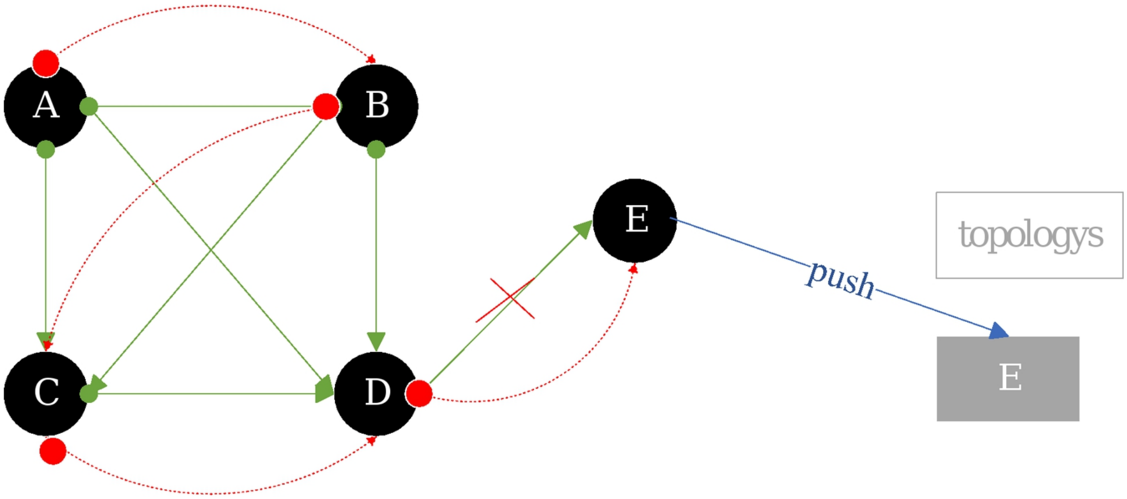
**1. The adjacency matrix is described above:**
The total number of vertices is a two-dimensional array size, if have value of the edge is 1 , otherwise no value of the edge is 0 .



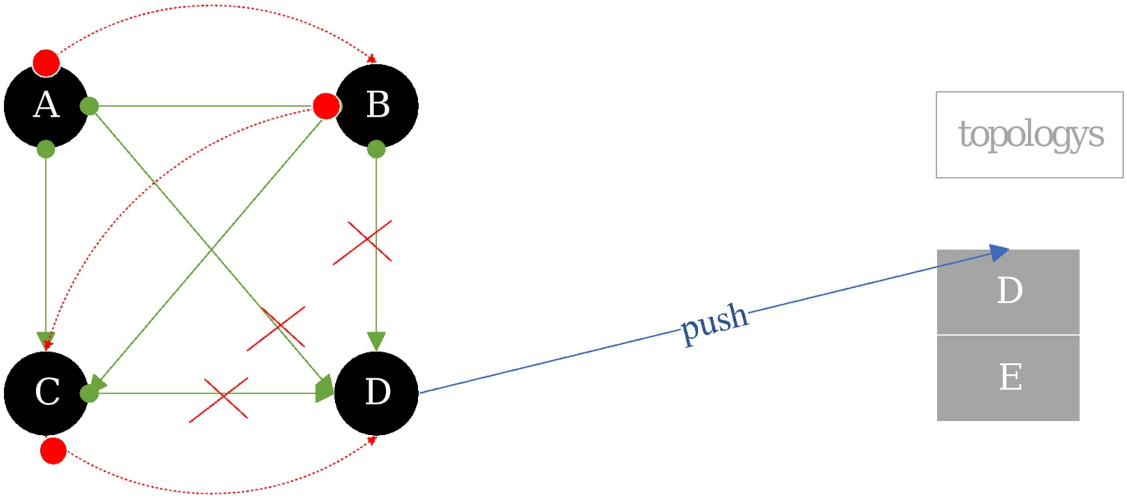
| | A | B | C | D | E |
|---|---|---|---|---|---|
| A | 0 | 1 | 1 | 1 | 0 |
| B | 0 | 0 | 1 | 1 | 0 |
| C | 0 | 0 | 0 | 1 | 0 |
| D | 0 | 0 | 0 | 0 | 1 |
| E | 0 | 0 | 0 | 0 | 0 |

**Topological sorting from vertex A : A -> B -> C -> D -> E**

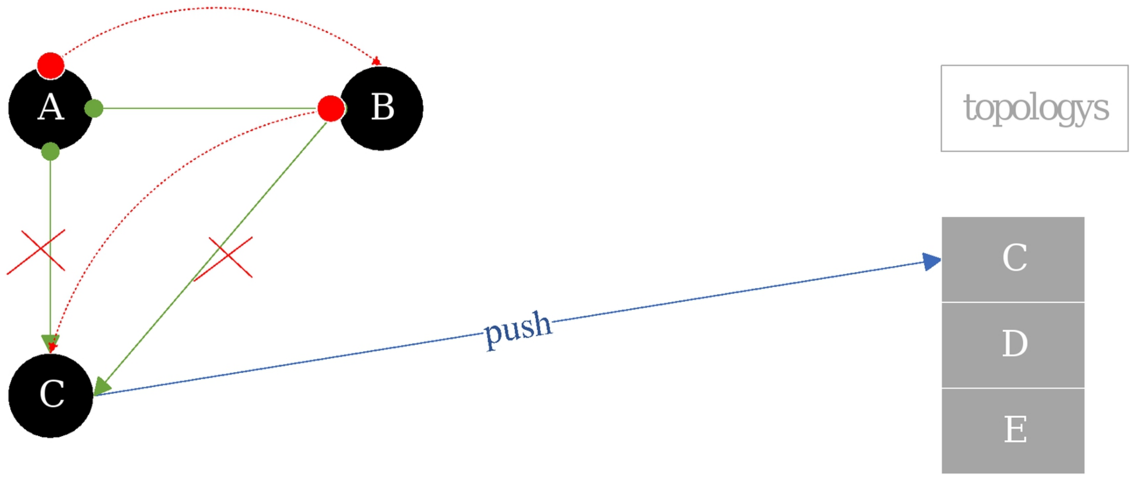
Find no successor vertices E then save to topologys, last E remove from the graph


A
B
C
D
E
push
topologys
E

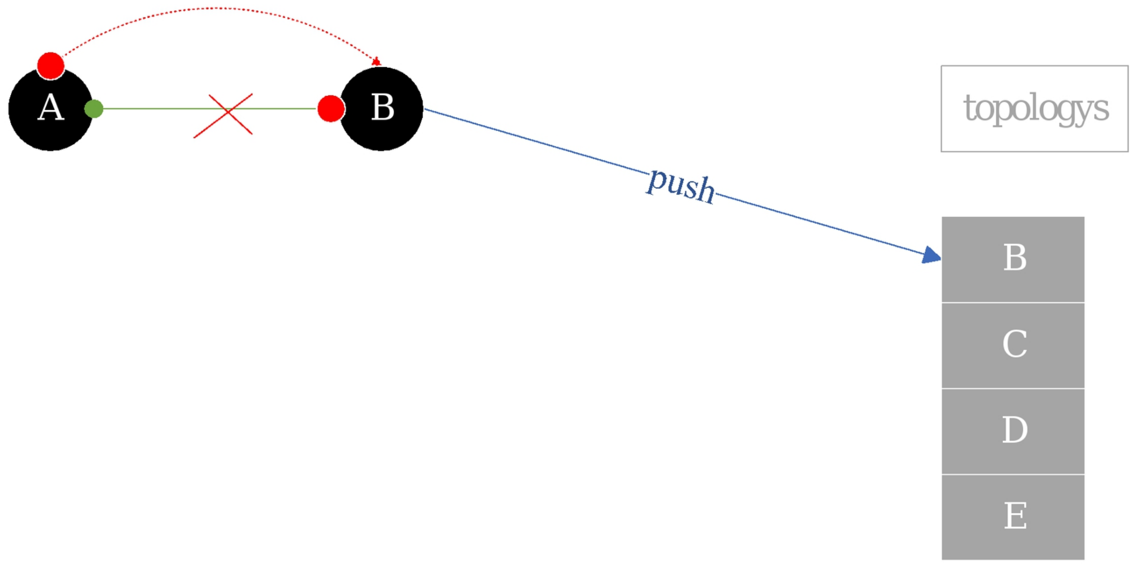
Find no successor vertices D then save to topologys, last D remove from the graph


A
B
C
D
push
topologys
D
E

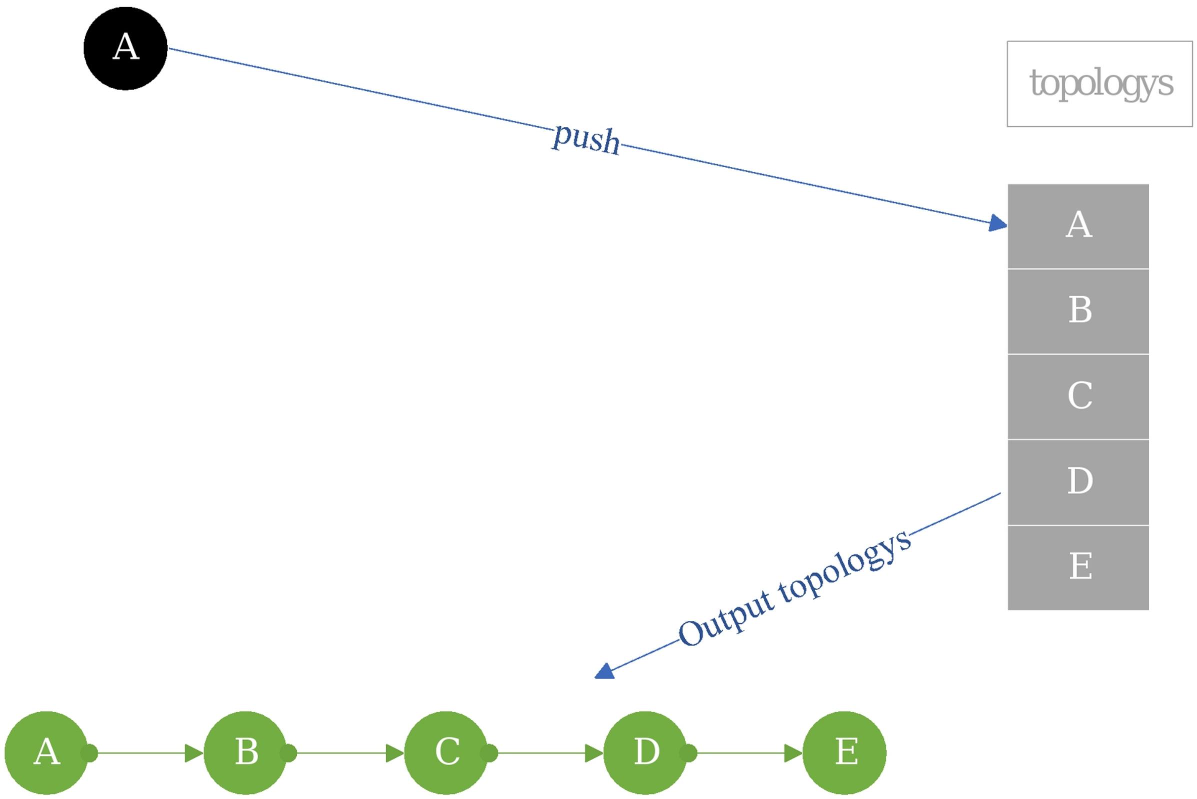
Find no successor vertices C then save to topologys, last C remove from the graph


A
B
C
push
topologys
C
D
E

Find no successor vertices C then save to topologys, last C remove from the graph


A
B
push
topologys
B
C
D
E

Find no successor vertices C then save to topologys, last C remove from the graph


A
push
topologys
A
B
C
D
E
Output topologys
A
B
C
D
E

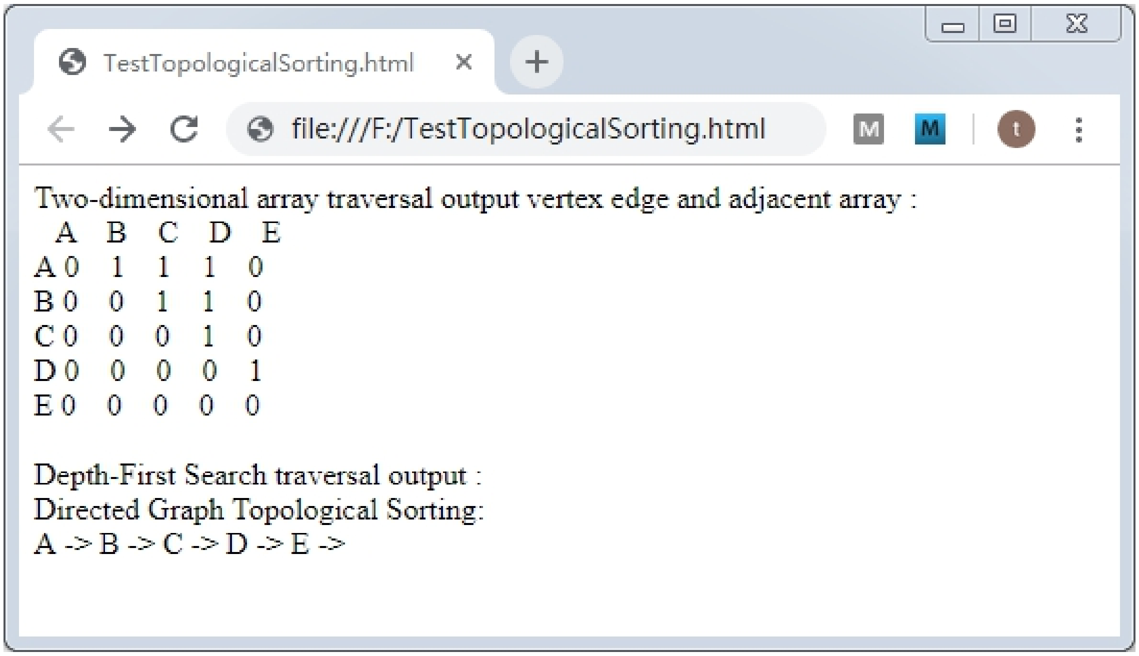
**Create a TestTopologicalSorting.html with Notepad**

```
<script type ="text/javascript" >
    class Vertex {
        constructor( data, visited){
            this.data = data;
            this.visited = visited; // Have you visited
        }

        getData() {
            return this.data;
        }
        setData( data) {
            this.data = data;
        }

        isVisited() {
            return this.visited;
        }
        setVisited( visited) {
            this.visited = visited;
        }
    }
```

```
class Graph {
    constructor( maxVertexSize){
        this.maxVertexSize = maxVertexSize; // Two-dimensional array size
        this.vertexs = new Array( maxVertexSize);
        this.topologys = new Array();
        this.adjacencyMatrix = new Array( maxVertexSize);
        for ( var i = 0 ; i < maxVertexSize; i++) {
            this.adjacencyMatrix[ i] = new Array();
            for ( var j = 0 ; j < maxVertexSize; j++) {
                this.adjacencyMatrix[ i][ j] = 0 ;
            }
        }
        this.size = 0 ;
    }

    addVertex( data) {
        var vertex = new Vertex( data, false );
        this.vertexs[ this.size] = vertex;
        this.size++;
    }

    // Add adjacent edges
    addEdge( from, to) {
        // A -> B  and B -> A are different
        this.adjacencyMatrix[ from][ to] = 1 ;
    }

    topologySort() {
        while ( this.size > 0 ) {
```

```
        var noSuccessorVertex = this.getNoSuccessorVertex(); // Get a no successor node
        if ( noSuccessorVertex == - 1 ) {
          document.write( "There is ring in Graph " );
          return ;
        }
        // Copy the deleted node to the sorted array
        this.topologys[ this.size - 1 ] = this.vertexs[ noSuccessorVertex];
        this.removeVertex( noSuccessorVertex); // Delete no successor node
      }
    }




    getNoSuccessorVertex() {
      var existSuccessor = false ;
      for ( var row = 0 ; row < this.size; row++) { // For each vertex
        existSuccessor = false ;
        //If the node has a fixed row, each column has a 1, indicating that the node has a successor, terminating the loop
        for ( var col = 0 ; col < this.size; col++) {
          if ( this.adjacencyMatrix[ row][ col] == 1 ) {
            existSuccessor = true ;
            break ;
          }
        }

        if (! existSuccessor) { // If the node has no successor, return its subscript
          return row;
        }
      }
      return - 1 ;
    }
    removeVertex( vertex) {
      if ( vertex != this.size - 1 ) { //If the vertex is the last element, the
end
        for ( var i = vertex; i < this.size - 1 ; i++) { // The vertices are
removed from the vertex array
          this.vertexs[ i] = this.vertexs[ i + 1 ];
        }

        for ( var row = vertex; row < this.size - 1 ; row++) { // move up a
row
          for ( var col = 0 ; col < this.size - 1 ; col++) {
            this.adjacencyMatrix[ row][ col] = this.adjacencyMatrix[
row + 1 ][ col];
          }
        }

        for ( var col = vertex; col < this.size - 1 ; col++) { // move left a
row
          for ( var row = 0 ; row < this.size - 1 ; row++) {
            this.adjacencyMatrix[ row][ col] = this.adjacencyMatrix[
row][ col + 1 ];
          }
        }
      }
      this.size--; // Decrease the number of vertices
    }



    // Clear reset
    clear() {
      for ( var i = 0 ; i < this.size; i++) {
        this.vertexs[ i]. setVisited( false );
      }
    }
    getAdjacencyMatrix() {
      return this.adjacencyMatrix;
    }

    getVertexs() {
      return this.vertexs;
    }

    getTopologys() {
      return this.topologys;
    }
  }

  ////////////////////testing/////////////////

  function printGraph( graph) {
    document.write( "Two-dimensional array traversal output vertex edge
and adjacent array :  <br>" );
    document.write( "&nbsp;&nbsp;&nbsp;" );
    for ( var i = 0 ; i < graph.getVertexs(). length; i++) {
      document.write( graph.getVertexs()[ i]. getData() +
"&nbsp;&nbsp;&nbsp;&nbsp;" );
    }
    document.write( "<br>" );

    for ( var i = 0 ; i < graph.getAdjacencyMatrix(). length; i++) {
      document.write( graph.getVertexs()[ i]. getData() + "  " );
      for ( var j = 0 ; j < graph.getAdjacencyMatrix(). length; j++) {
        document.write( graph.getAdjacencyMatrix()[ i][ j] +
"&nbsp;&nbsp;&nbsp;&nbsp;" );
      }
      document.write( "<br>" );
    }
  }
  var graph = new Graph( 5 );

  graph.addVertex( "A" );
  graph.addVertex( "B" );
  graph.addVertex( "C" );
  graph.addVertex( "D" );
  graph.addVertex( "E" );

  graph.addEdge( 0 , 1 );
  graph.addEdge( 0 , 2 );
  graph.addEdge( 0 , 3 );
  graph.addEdge( 1 , 2 );
  graph.addEdge( 1 , 3 );
  graph.addEdge( 2 , 3 );
  graph.addEdge( 3 , 4 );

  // Two-dimensional array traversal output vertex edge and adjacent array
  printGraph( graph);

  document.write( "<br>Depth-First Search traversal output : <br>" );
  document.write( "Directed Graph Topological Sorting:<br>" );
  graph.topologySort();
  for ( var i= 0 ; i< graph.getTopologys(). length ; i++){
    document.write( graph.getTopologys()[ i]. getData()+ " -> " );
  }

</script>
```

## Result:


TestTopologicalSorting.html
file:///F:/TestTopologicalSorting.html

```
Two-dimensional array traversal output vertex edge and adjacent array :
  A   B   C   D   E
A 0   1   1   1   0
B 0   0   1   1   0
C 0   0   0   1   0
D 0   0   0   0   1
E 0   0   0   0   0

Depth-First Search traversal output :
Directed Graph Topological Sorting:
A -> B -> C -> D -> E ->
```

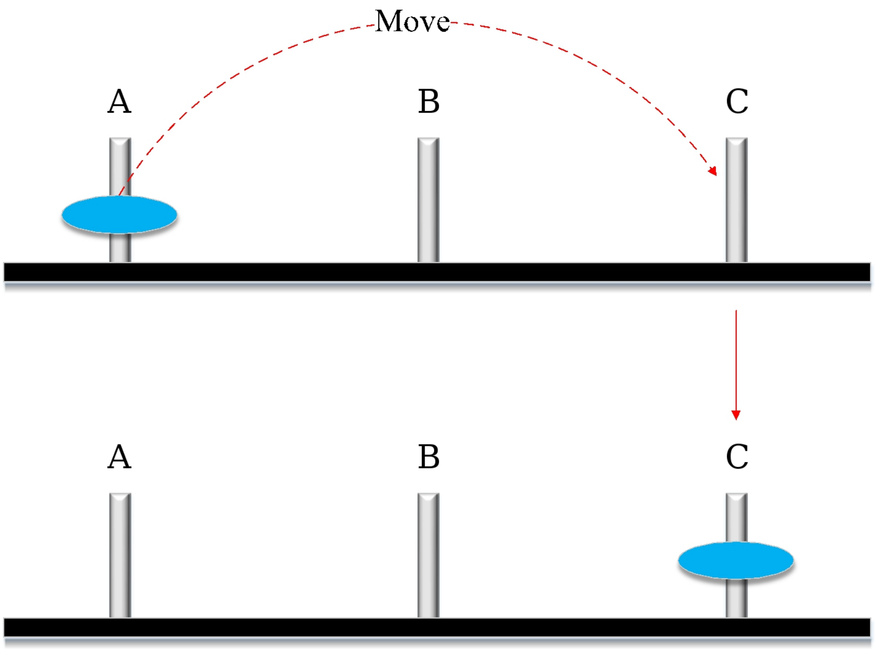
# Towers of Hanoi

The Hanoi Tower that a Frenchman M. Claus (Lucas) from Thailand to France in 1883. Hanoi Tower which is supported by three diamond Pillars. At the beginning, God placed 64 gold discs from top to bottom on the first Pillar. God ordered the monks to move all gold discs from the first Pillar to the third Pillar. The principle of large plates under small plates during the handling process. If only one plate is moved daily, the tower will be destroyed. when all the discs are moved that is the end of the world.

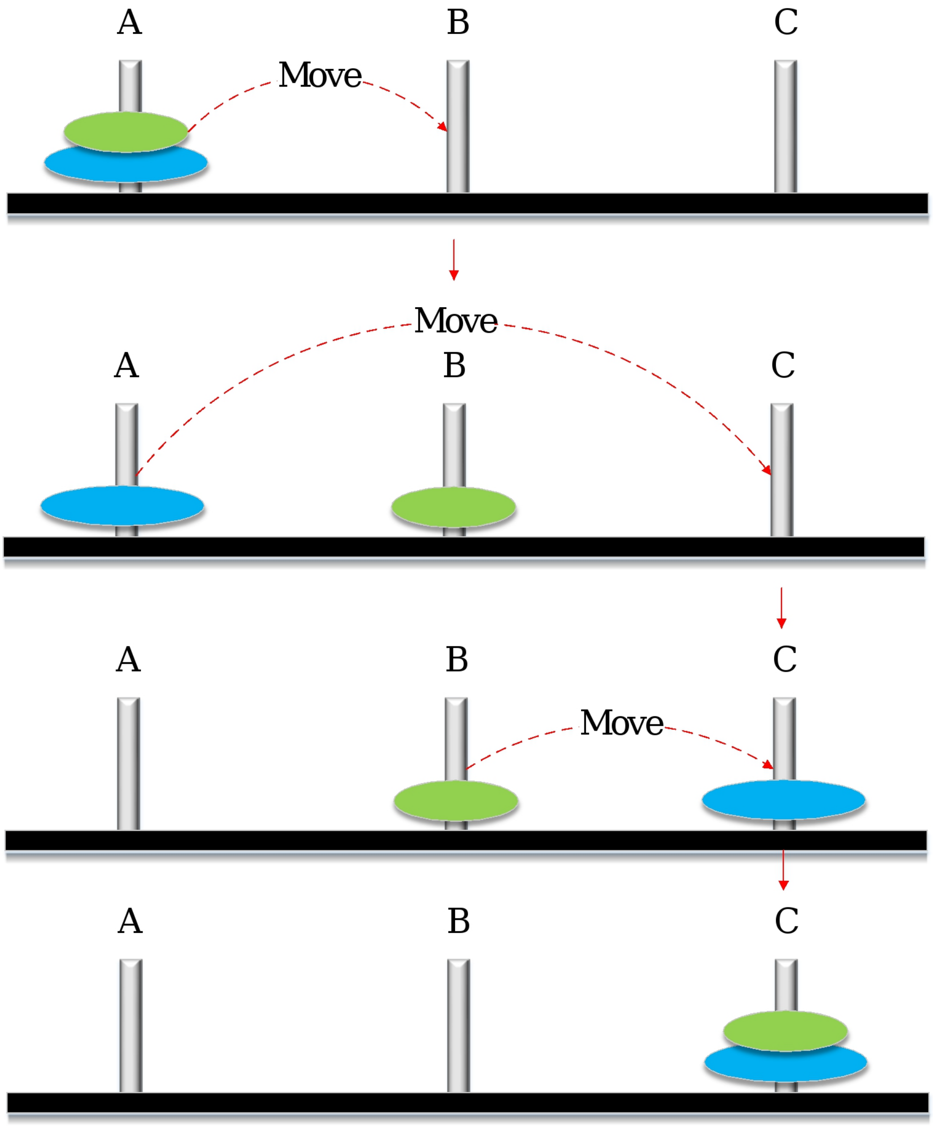
## Let's turn this story into an algorithm:

Mark the three columns as ABC.
1. If there is only one disc, move it directly to C (A->C).
2. When there are two discs, use B as an auxiliary (A->B, A->C,B->C).
3. If there are more than two discs, use B as an auxiliary(A->B, A->C,B->C), and continue to recursive process.

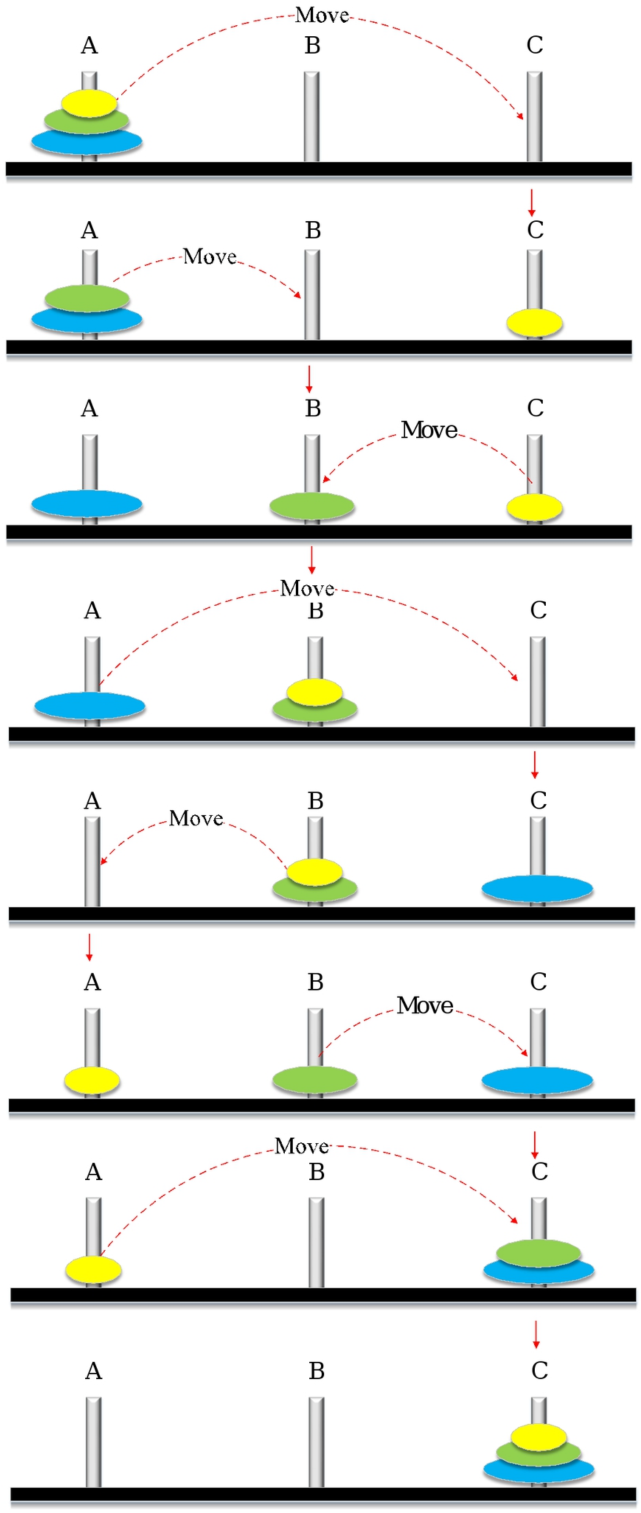
## 1. If there is only one disc, move it directly to C (A->C).



**2. When there are two discs, use B as an auxiliary (A->B, A->C,B->C).**


A
B
C
Move
Move
A
B
C
A
B
C
Move
A
B
C

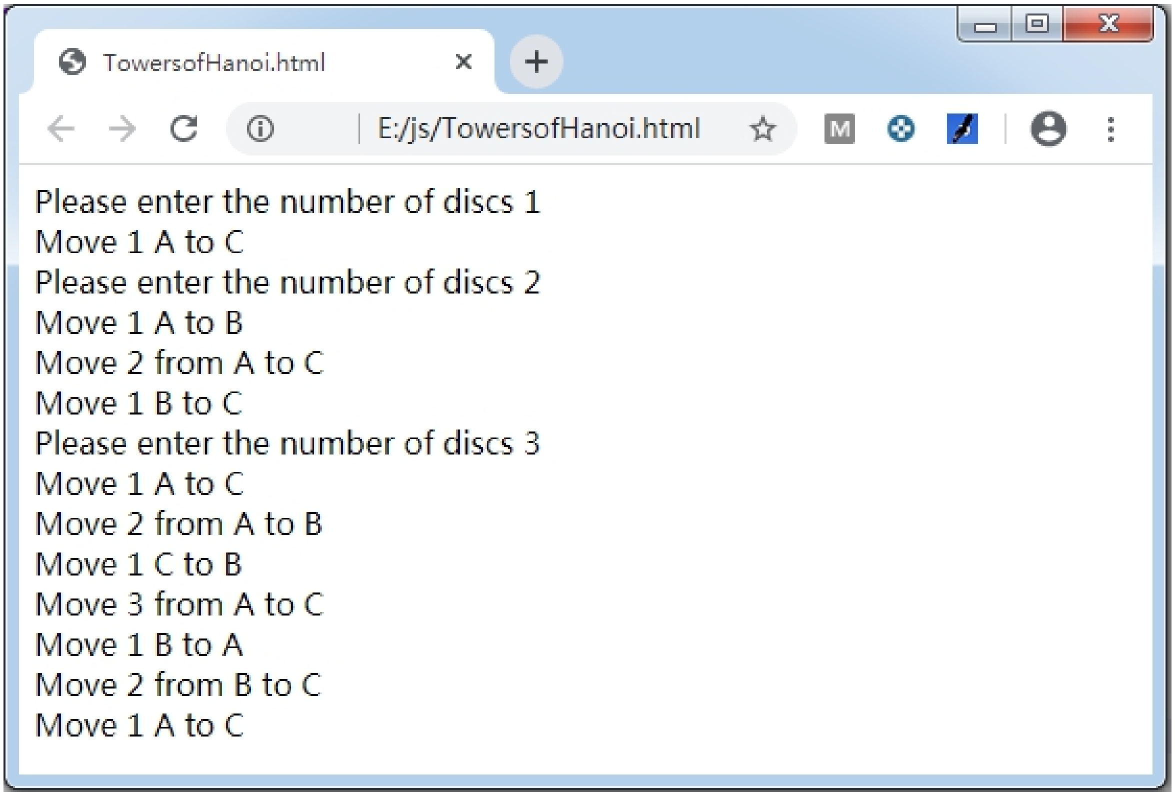
**3. If more than two discs, use B as an auxiliary, and continue to recursive process.**


Move
A
B
C
A
B
C
Move
A
B
C
Move
Move
A
B
C
A
B
C
Move
A
B
C
Move
Move
A
B
C
A
B
C

**1. Create a TowersofHanoi.html with Notepad and open it in your browser.**

```
<script type ="text/javascript" >
   function hanoi( n, A, B, C) {
      if ( n == 1 ) {
         document.write( "Move " + n + " " + A + "  to " + C + "<br>" );
      } else {
         hanoi( n - 1 , A, C, B); // Move the n-1th disc on the A  through C to  B
         document.write( "Move " + n + " from " + A + "  to " + C + "<br>" );
         hanoi( n - 1 , B, A, C); //Move the n-1th disc on the B  through  A to  C
      }
   }

   //////////////////////testing////////////////////
   document.write( "Please enter the number of discs 1 <br>" );
   var n = 1 ;
   hanoi( n, 'A' , 'B' , 'C' );

   document.write( "Please enter the number of discs 2 <br>" );
   var n = 2 ;
   hanoi( n, 'A' , 'B' , 'C' );

   document.write( "Please enter the number of discs 3 <br>" );
   var n = 3 ;
   hanoi( n, 'A' , 'B' , 'C' );
</script>
```

**Result:**


TowersofHanoi.html
E:/js/TowersofHanoi.html

```
Please enter the number of discs 1
Move 1 A to C
Please enter the number of discs 2
Move 1 A to B
Move 2 from A to C
Move 1 B to C
Please enter the number of discs 3
Move 1 A to C
Move 2 from A to B
Move 1 C to B
Move 3 from A to C
Move 1 B to A
Move 2 from B to C
Move 1 A to C
```

# Fibonacci

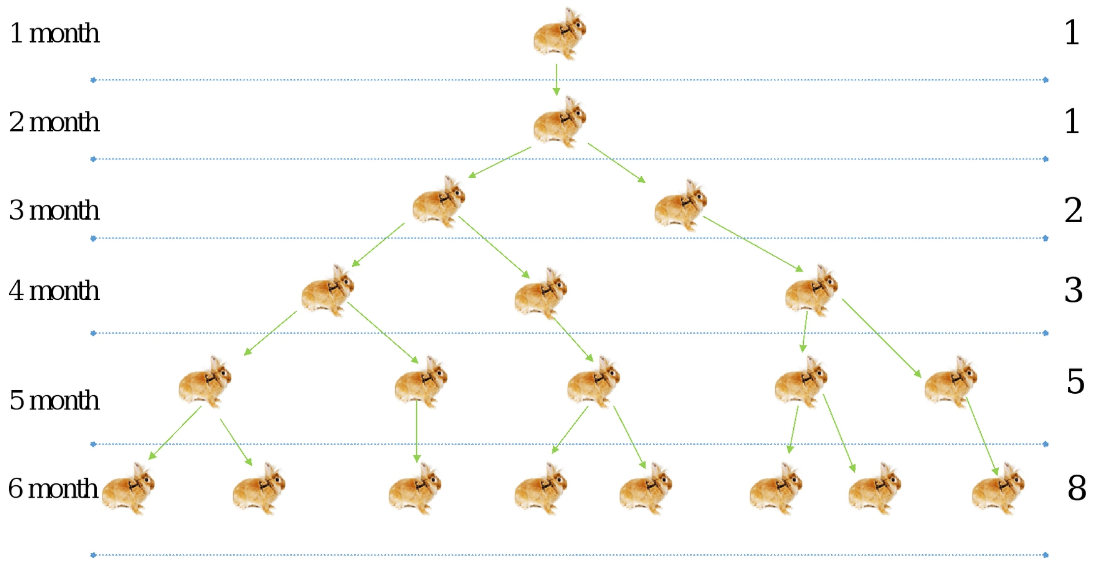
Fibonacci : a European mathematician in the 1200s, in his writings: "If there is a rabbit

After a month can birth to a newborn rabbit. At first there was only 1 rabbit, after one month still 1 rabbit. after two month 2 rabbit, and after three months there are 3 rabbit ......

for example: 1, 1, 2, 3, 5, 8, 13 ...

## Fibonacci definition:

```
if n = 0, 1
    fn = n
if n > 1
    fn = fn-1 + fn-2
```

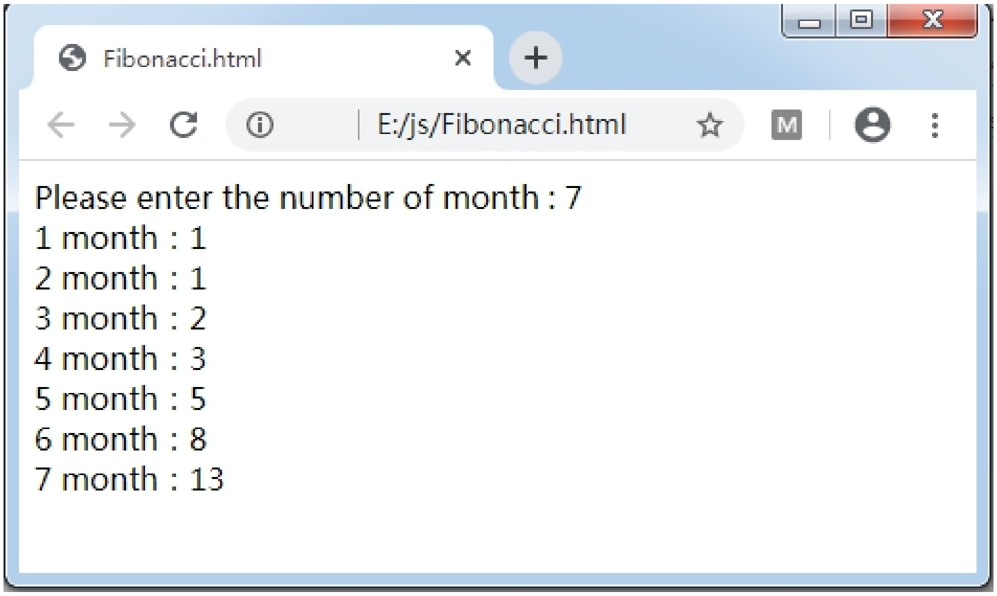


1 month
1
2 month
1
3 month
2
4 month
3
5 month
5
6 month
8

### 1. Create a Fibonacci.html with Notepad and open it in your browser.

```
<script type ="text/javascript" >
  function fibonacci( n) {
     if ( n == 1 || n == 2 ) {
        return 1 ;
     } else {
        return fibonacci( n - 1 ) + fibonacci( n - 2 );
     }
  }

  ////////////////////testing/////////////////
  document.write( "Please enter the number of month : 7 <br>" );
  var number = 7 ;

  for ( var i = 1 ; i <= number; i++) {
     document.write( i + " mont h ：  " + fibonacci( i) + "<br>" );
  }
</script>
```

**Result:**

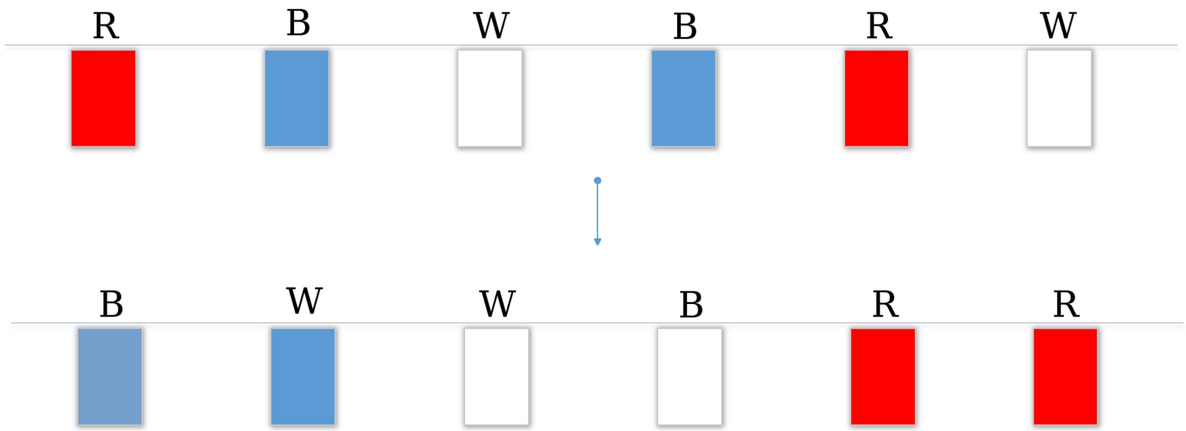


```
Please enter the number of month : 7
1 month : 1
2 month : 1
3 month : 2
4 month : 3
5 month : 5
6 month : 8
7 month : 13
```

# Dijkstra

The tricolor flag was originally raised by E.W. Dijkstra, who used the Dutch national flag (Dijkstra is Dutch).

Suppose there is a rope with red, white, and blue flags. At first all the flags on the rope are not in order. You need to arrange them in the order of blue -> white -> red. How to move them with the least times. you just only do this on the rope, and only swap two flags at a time.



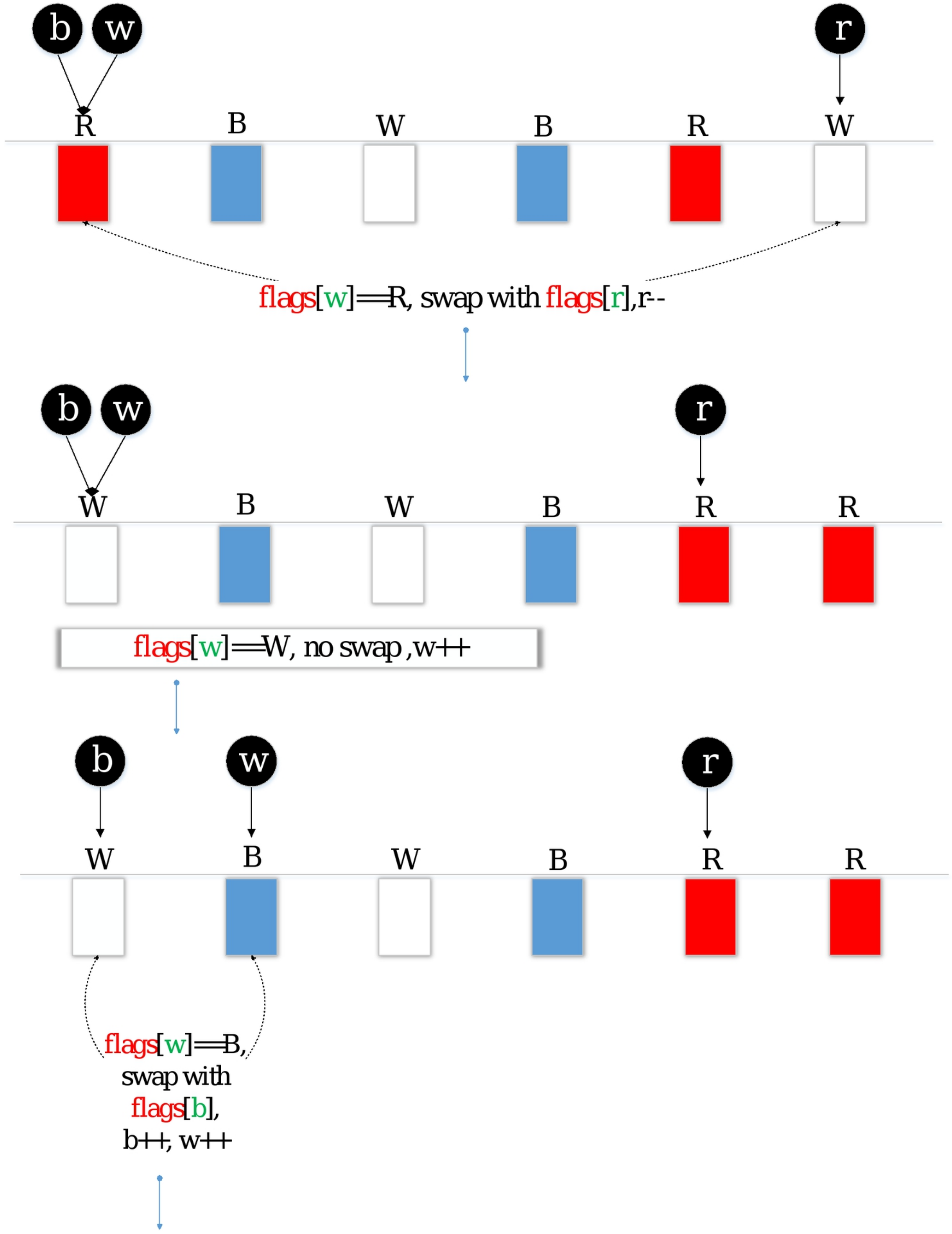
**Solution:**

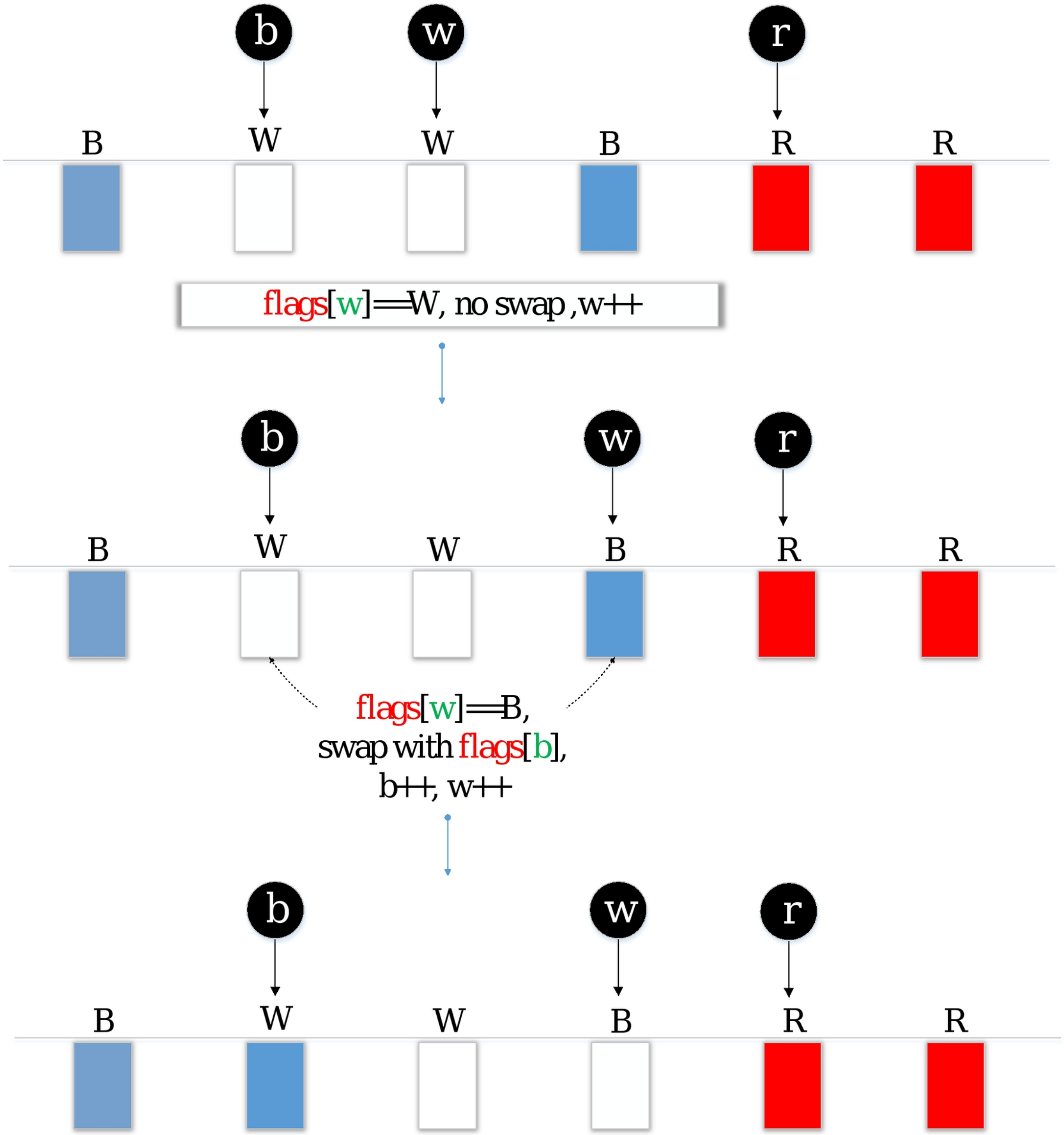
Use the char arrays to store the flags . For example, b , w , and r indicate the position of **blue** , **white** and **red** flags. The beginning of b and w is 0 of the array, and r is at the end of the array.

(1) If the position of w is a blue flag, flags [w ] exchange with flags [b ]. And whiteIndex and b is moved backward by 1.
(2) If the position of w is a white flag, w moves backward by 1.
(3) If the position of w is a red flag, flags [w ] exchange with flags [r ]. r moves forward by 1.

In the end, the flags in front of b are all blue, and the flags behind r are all red.

## Graphic Solution

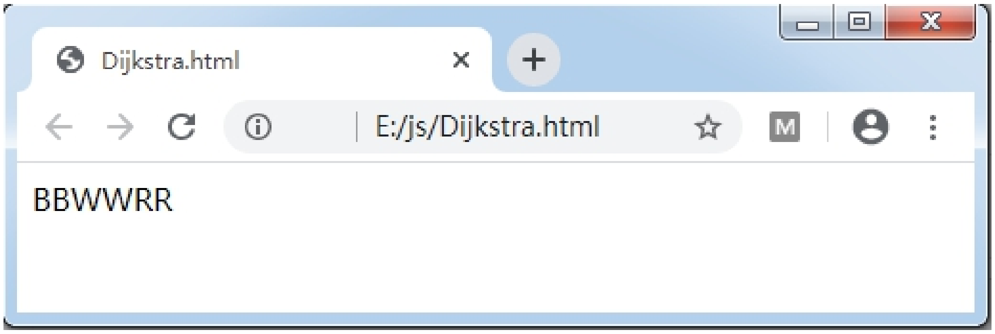

b
w
r
R
B
W
B
R
W
flags[w]==R, swap with flags[r],r--
b
w
r
W
B
W
B
R
R
flags[w]==W, no swap ,w++
b
w
r
W
B
W
B
R
R
flags[w]==B,
swap with
flags[b],
b++, w++


b
w
r
B
W
W
B
R
R
flags[w]==W, no swap ,w++
b
w
r
B
W
W
B
R
R
flags[w]==B,
swap with flags[b],
b++, w++
b
w
r
B
W
W
B
R
R

**1. Create a Dijkstra.html with Notepad and open it in your browser.**

```
<script type ="text/javascript" >
  function dijkstra( flags) {
    var b = 0 , w = 0 , r = flags.length - 1 ;
    var count = 0 ;
    while ( w <= r) {
      if ( flags[ w] == 'W' ) {
        w++;
      } else if ( flags[ w] == 'B' ) {
        var temp = flags[ w];
        flags[ w] = flags[ b];
        flags[ b] = temp;
        w++;
        b++;
        count++;
      } else if ( flags[ w] == 'R' ) {
        var m = flags[ w];
        flags[ w] = flags[ r];
        flags[ r] = m;
        r--;
        count++;
      }
    }
  }

  ////////////////////testing/////////////////
  var flags = [ 'R' , 'B' , 'W' , 'B' , 'R' , 'W' ];
  dijkstra( flags);
  for ( var i = 0 ; i < flags.length; i++) {
    document.write( flags[ i]);
  }
</script>
```

**Result:**



Dijkstra.html
E:/js/Dijkstra.html

```
BBWWRR
```

# Mouse Walking Maze

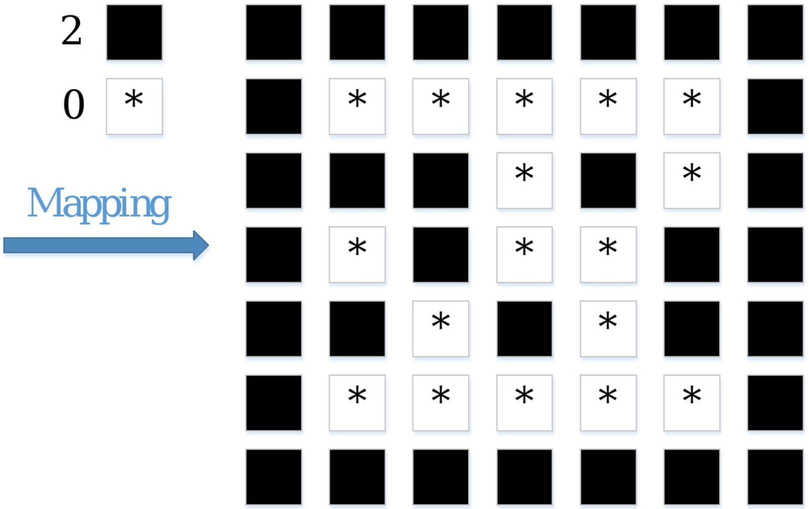
Mouse Walking Maze is a basic type of recursive solution. We use 2 to represent the wall in a two-dimensional array , and use 1 to represent the path of the mouse, and try to find the path from the entrance to the exit.

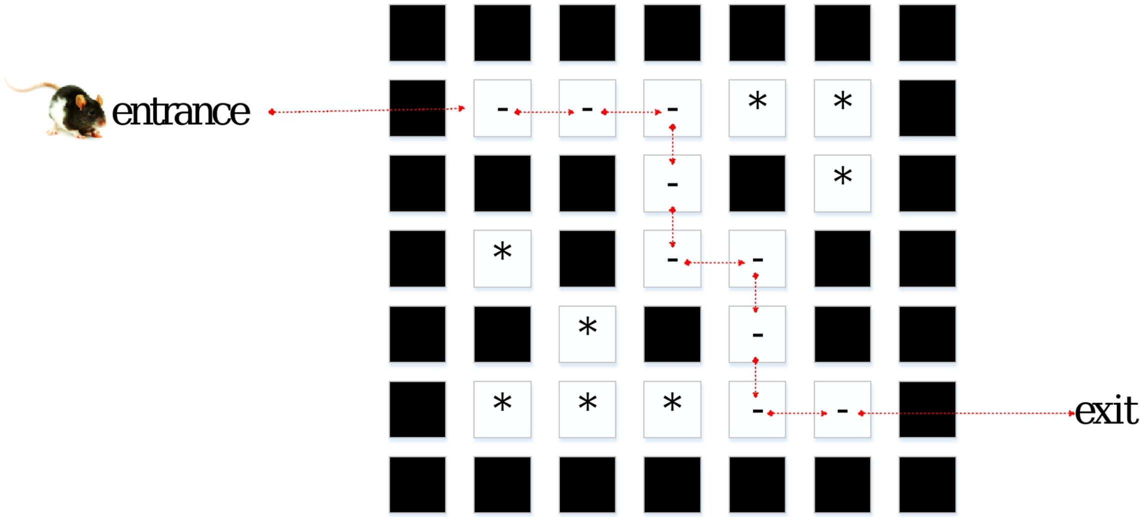
**Solution:**

The mouse moves in four directions: up, left, down, and right . if hit the wall go back and select the next forward direction, so test the four directions in the array until mouse reach the exit.

**Graphic Solution**

```
{ 2, 2, 2, 2, 2, 2, 2 }
{ 2, 0, 0, 0, 0, 0, 2 }
{ 2, 2, 2, 0, 2, 0, 2 }
{ 2, 0, 2, 0, 0, 2, 2 }
{ 2, 2, 0, 2, 0, 2, 2 }
{ 2, 0, 0, 0, 0, 0, 2 }
{ 2, 2, 2, 2, 2, 2, 2 }
```



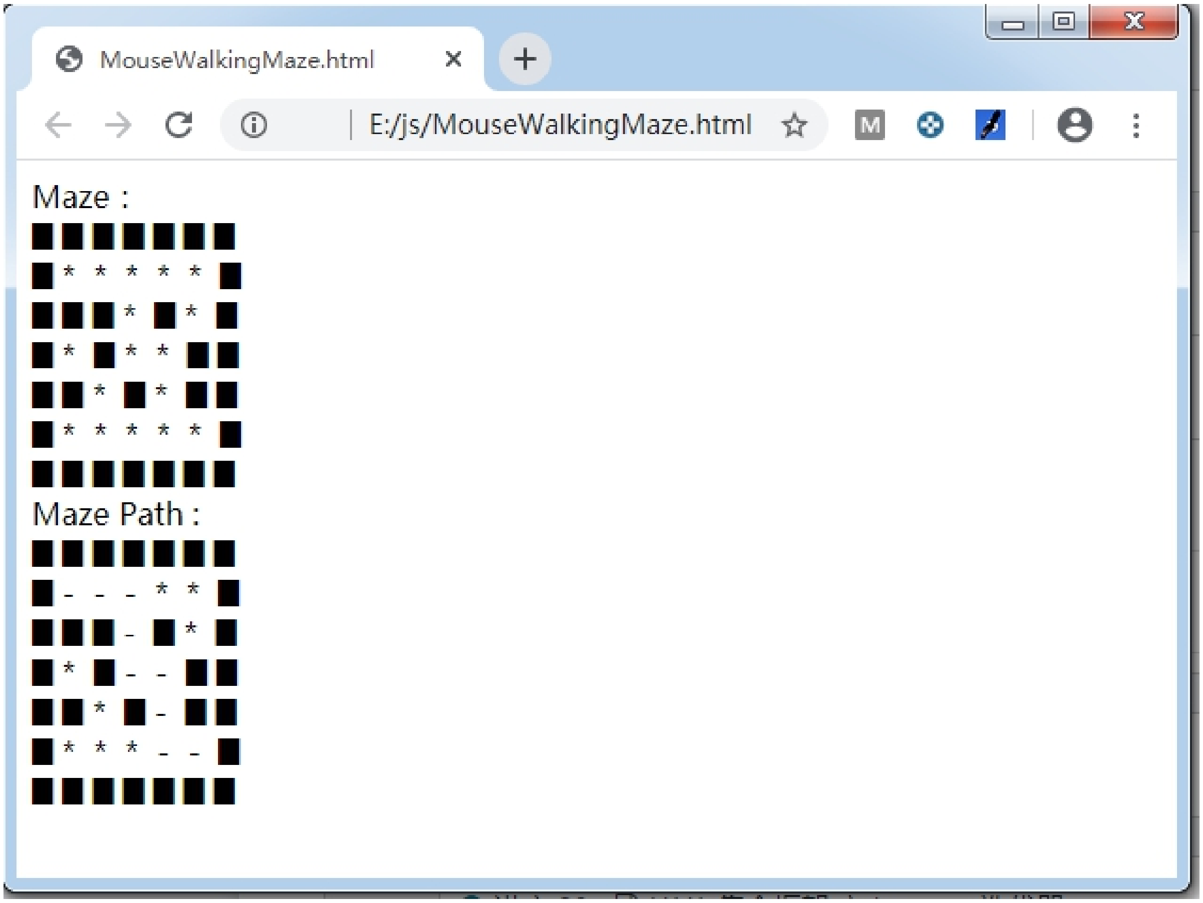


**1. Create a MouseWalkingMaze.html with Notepad and open it in your browser.**

```
<script type ="text/javascript" >
   var maze = [
      [ 2 , 2 , 2 , 2 , 2 , 2 , 2 ],
      [ 2 , 0 , 0 , 0 , 0 , 0 , 2 ],
      [ 2 , 2 , 2 , 0 , 2 , 0 , 2 ],
      [ 2 , 0 , 2 , 0 , 0 , 2 , 2 ],
      [ 2 , 2 , 0 , 2 , 0 , 2 , 2 ],
      [ 2 , 0 , 0 , 0 , 0 , 0 , 2 ],
      [ 2 , 2 , 2 , 2 , 2 , 2 , 2 ]
      ];

   var startI = 1 ;
   var startJ = 1 ;
   var endI = 5 ;
   var endJ = 5 ;
   var success = 0 ;

   //The mouse moves in four directions: up, left, down, and right. if hit the wall go back and select the next forward direction
   function visit( i, j) {
      maze[ i][ j] = 1 ;
      if ( i == endI && j == endJ) {
         success = 1 ;
      }
      if ( success != 1 && maze[ i][ j + 1 ] == 0 )
         visit( i, j + 1 );
      if ( success != 1 && maze[ i + 1 ][ j] == 0 )
         visit( i + 1 , j);
      if ( success != 1 && maze[ i][ j - 1 ] == 0 )
         visit( i, j - 1 );
      if ( success != 1 && maze[ i - 1 ][ j] == 0 )
         visit( i - 1 , j);
      if ( success != 1 )
      maze[ i][ j] = 0 ;
   return success;
}






//////////////////////testing////////////////////
document.write( "Maz e ：  <br>" );
for ( var i = 0 ; i < 7 ; i++) {
   for ( var j = 0 ; j < 7 ; j++) {
      if ( maze[ i][ j] == 2 ) {
         document.write( "█&nbsp;" );
      } else {
         document.write( "*&nbsp;&nbsp;" );
      }
   }
   document.write( "<br>" );
}

if ( visit( startI, startJ) == 0 ) {
   document.write( "No exit found <br>" );
} else {
   document.write( "Maze Path : <br>" );
   for ( var i = 0 ; i < 7 ; i++) {
      for ( var j = 0 ; j < 7 ; j++) {
         if ( maze[ i][ j] == 2 ) {
            document.write( "█&nbsp;" );
         } else if ( maze[ i][ j] == 1 ) {
            document.write( "-&nbsp;&nbsp;" );
         } else {
            document.write( "*&nbsp;&nbsp;" );
         }
      }
      document.write( "<br>" );
```

```
        }
    }
</script>
```

## Result:


MouseWalkingMaze.html
E:/js/MouseWalkingMaze.html

```
Maze :
█ █ █ █ █ █ █
█ * * * * * █
█ █ █ * █ * █
█ * █ * * █ █
█ █ * █ * █ █
█ * * * * * █
█ █ █ █ █ █ █
Maze Path :
█ █ █ █ █ █ █
█ - - - * * █
█ █ █ - █ * █
█ * █ - - █ █
█ █ * █ - █ █
█ * * * - - █
█ █ █ █ █ █ █
```

# Eight Coins

There are eight coins with the same appearance, one is a counterfeit coin, and the weight of counterfeit coin is different from the real coin, but it is unknown whether the counterfeit coin is lighter or heavier than the real coin. Please design a efficient algorithm to detect this counterfeit coin.

**Solution:**

Take six a, b, c, d, e, f from eight coins, and put three to the balance for comparison. Suppose a, b, c are placed on one side, and d, e, f are placed on the other side.

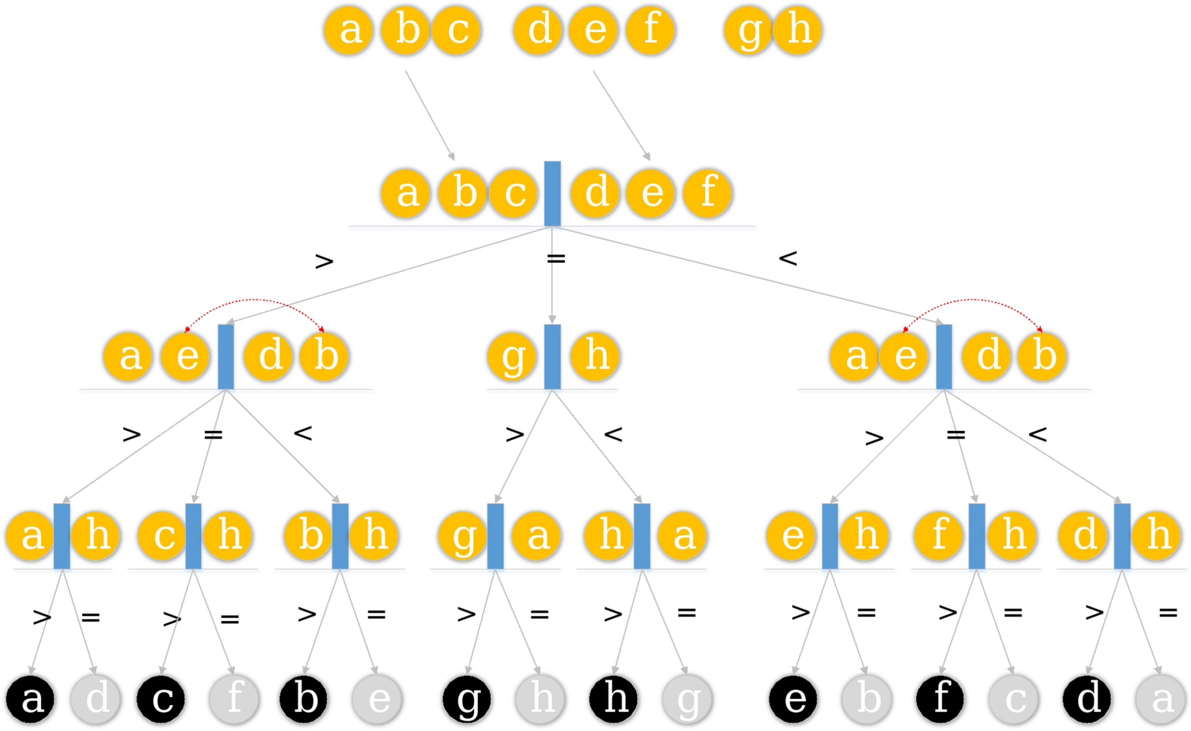
1. $a + b + c > d + e + f$
2. $a + b + c = d + e + f$
3. $a + b + c < d + e + f$

If $a + b + c > d + e + f$, there is a counterfeit coin in one of the six coins, and g, h are real coins. At this time, one coin can be removed from both side. Suppose that c and f are removed. At the same time, one coin at each side is replaced. Suppose the coins b and e are interchanged, and then the second comparison. There are also three possibilities:

1. $a + e > d + b$: the counterfeit currency must be one of a, d . as long as we compare a real currency h with a , we can find the counterfeit currency. If $a > h$ , a is a heavier counterfeit currency; if $a = h$ , d is a lighter counterfeit currency.
2. $a + e = d + b$: the counterfeit currency must be one of c, f, and the real coin h is compared with c . If $c > h$ , c is a heavier counterfeit currency; if $c = h$ , then f is a lighter counterfeit currency.
3. $a + e < d + b$: one of b or e is a counterfeit coin , Also use the real coin h to compare with b, if $b > h$, then b is a heavier counterfeit currency; if $b = h$, then e is a lighter counterfeit currency;

**Graphic Solution**

● **Heavy** ○ **Light**

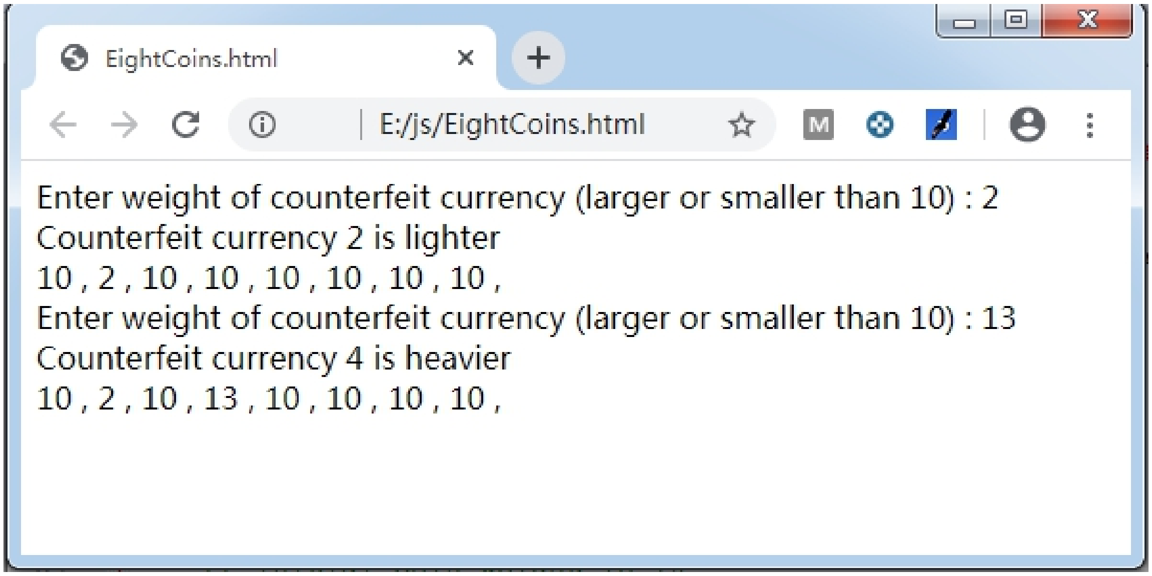

a b c d e f g h
a b c d e f
>
=
<
a e d b
g h
a e d b
> = <
> <
> = <
a h c h b h g a h a e h f h d h
> = > = > = > = > = > = > = > =
a d c f b e g h h g e b f c d a

**1. Create a EightCoins.html with Notepad and open it in your browser.**

```
<script type ="text/javascript" >
   function compare( coins, i, j, k) { //coin[k] true,coin[i]>coin[j]
      if ( coins[ i] > coins[ k]) //coin[i]>coin[j]&&coin[i]>coin[k] ----->coin[i] is a heavy counterfeit coin
         document.write( "\nCounterfeit currency " + ( i + 1 ) + " is heavier <br>" );
      else //coin[j] is a light counterfeit coin
         document.write( "\nCounterfeit currency " + ( j + 1 ) + " is lighter <br>" );
   }

   function eightcoins( coins) {
      if ( coins[ 0 ] + coins[ 1 ] + coins[ 2 ] == coins[ 3 ] + coins[ 4 ] + coins[ 5 ]){ //(a+b+c)==(d+e+f)
         if ( coins[ 6 ] > coins[ 7 ]) //g>h?(g>a?g:a):(h>a?h:a)
            compare( coins, 6 , 7 , 0 );
         else //h>g?(h>a?h:a):(g>a?g:a)
            compare( coins, 7 , 6 , 0 );
      } else if ( coins[ 0 ] + coins[ 1 ] + coins[ 2 ] > coins[ 3 ] + coins[ 4 ] + coins[ 5 ]){ //(a+b+c)>(d+e+f)
         if ( coins[ 0 ] + coins[ 3 ] == coins[ 1 ] + coins[ 4 ]) //(a+e)==(d+b)
            compare( coins, 2 , 5 , 0 );
         else if ( coins[ 0 ] + coins[ 3 ] > coins[ 1 ] + coins[ 4 ]) //(a+e)>(d+b)
            compare( coins, 0 , 4 , 1 );
         if ( coins[ 0 ] + coins[ 3 ] < coins[ 1 ] + coins[ 4 ]) //(a+e)<(d+b)
            compare( coins, 1 , 3 , 0 );
      } else if ( coins[ 0 ] + coins[ 1 ] + coins[ 2 ] < coins[ 3 ] + coins[ 4 ] + coins[ 5 ]) { //(a+b+c)<(d+e+f)
         if ( coins[ 0 ] + coins[ 3 ] == coins[ 1 ] + coins[ 4 ]) //(a+e)>(d+b)
            compare( coins, 5 , 2 , 0 );
         else if ( coins[ 0 ] + coins[ 3 ] > coins[ 1 ] + coins[ 4 ]) //(a+e)>(d+b)
            compare( coins, 3 , 1 , 0 );
         if ( coins[ 0 ] + coins[ 3 ] < coins[ 1 ] + coins[ 4 ]) //(a+e)<(d+b)
            compare( coins, 4 , 0 , 1 );
      }
   }














   /////////////////////testing///////////////////
   var coins = new Array();
   // Initial coin weight is 10
   for ( var i = 0 ; i < 8 ; i++)
      coins[ i] = 10 ;

   document.write( "Enter weight of counterfeit currency  (larger or smaller
than 10) : 2<br>" );
   coins[ 1 ] = 2 ;
   eightcoins( coins);
   for ( var i = 0 ; i < 8 ; i++)
      document.write( coins[ i]+ " , " );

   document.write( "<br>" );

   document.write( "Enter weight of counterfeit currency  (larger or smaller
than 10) : 13<br>" );
   coins[ 3 ] = 13 ;
   eightcoins( coins);
   for ( var i = 0 ; i < 8 ; i++)
      document.write( coins[ i]+ " , " );
```

```
</script>
```

**Result:**


EightCoins.html
E:/js/EightCoins.html

```
Enter weight of counterfeit currency (larger or smaller than 10) : 2
Counterfeit currency 2 is lighter
10 , 2 , 10 , 10 , 10 , 10 , 10 , 10 ,
Enter weight of counterfeit currency (larger or smaller than 10) : 13
Counterfeit currency 4 is heavier
10 , 2 , 10 , 13 , 10 , 10 , 10 , 10 ,
```

# Josephus Problem

There are 9 Jewish hid in a hole with Josephus and his friends . The 9 Jews decided to die rather than be caught by the enemy, so they decided In a suicide method, 11 people are arranged in a circle, and the first person reports the number. After each number is reported to the third person, the person must commit suicide. Then count again from the next one until everyone commits suicide.

But Josephus and his friends did not want to obey. Josephus asked his friends to pretend to obey, and he arranged the friends with himself. In the 2th and 7st positions, they escaped this death game.

**Solution:**

As long as the array is treated as a ring.

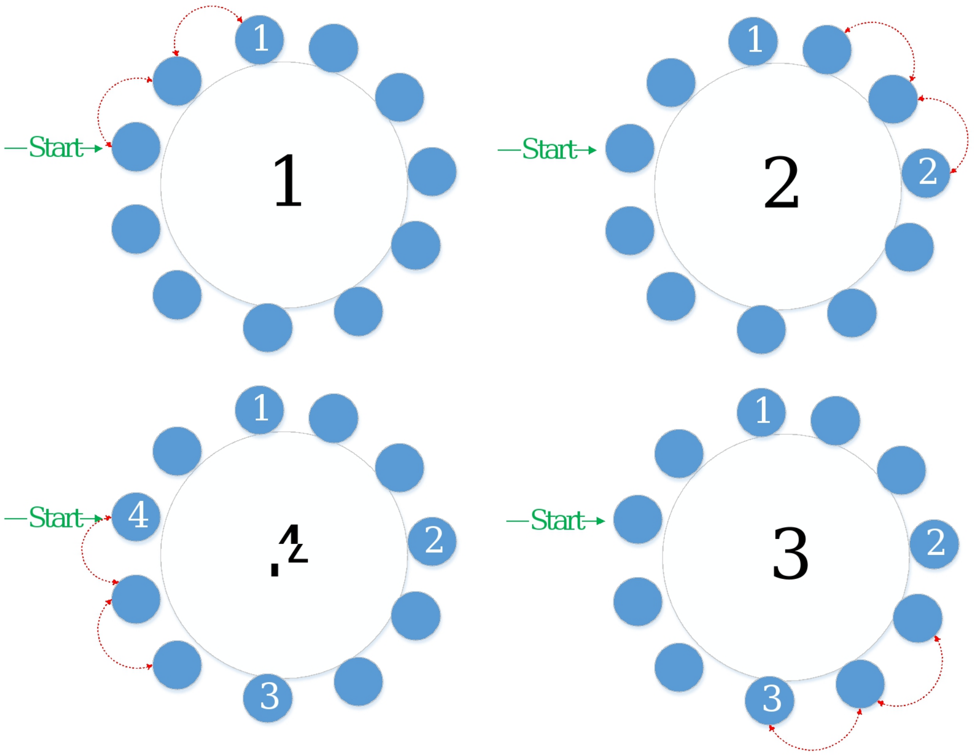
Fill in a count for each dataless area, until the count reaches 11, and then list the array from index 1, you can know that each suicide order in this position is the Joseph's position.

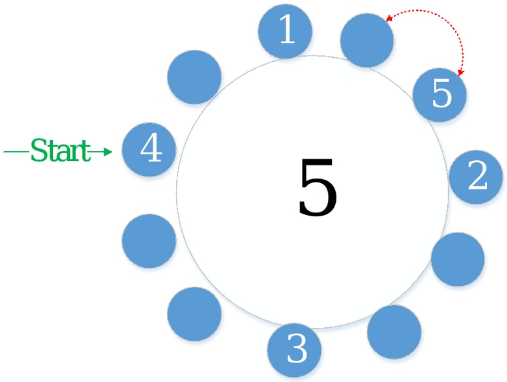
The 11-person position is as follows:

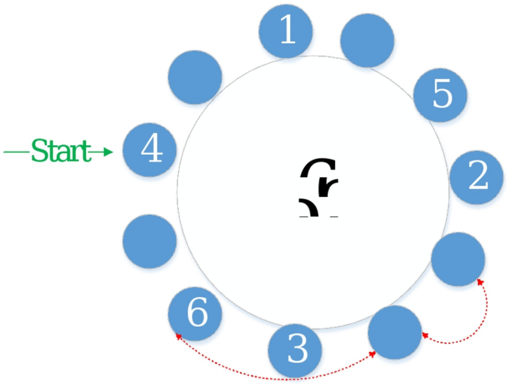
4 10 1 7 5 2 11 9 3 6 8

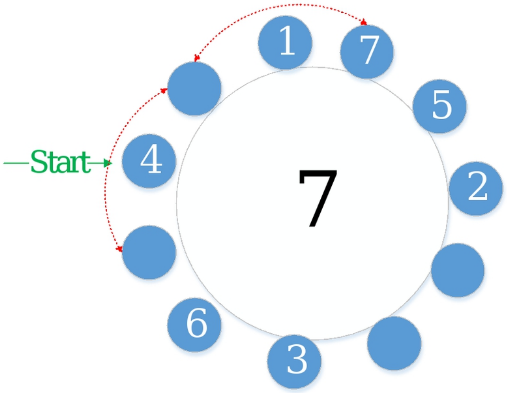
From the above, the last two suicide was in the 31st and 16th position. The previous one

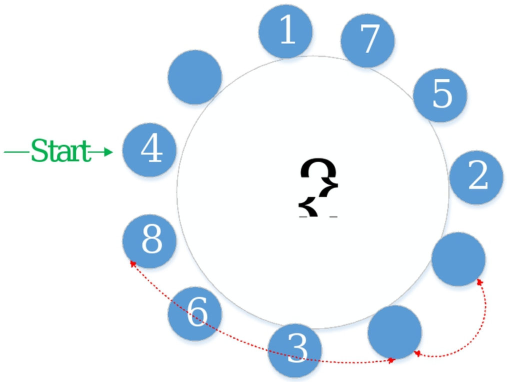
Everyone died, so they didn't know that Joseph and his friends didn't follow the rules of the game.

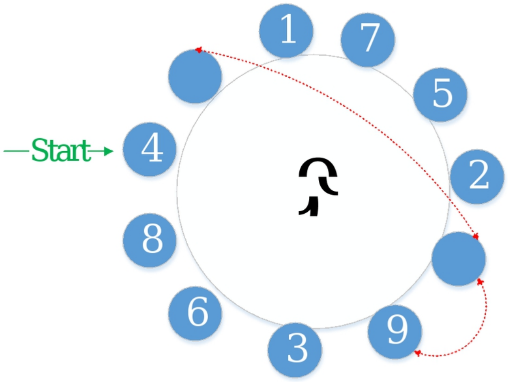

Start
Start
Start
Start

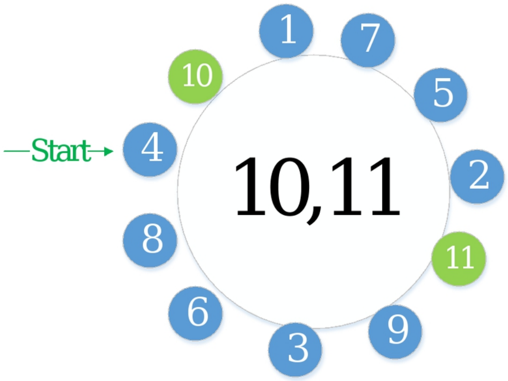

Start


Start


Start


Start


Start

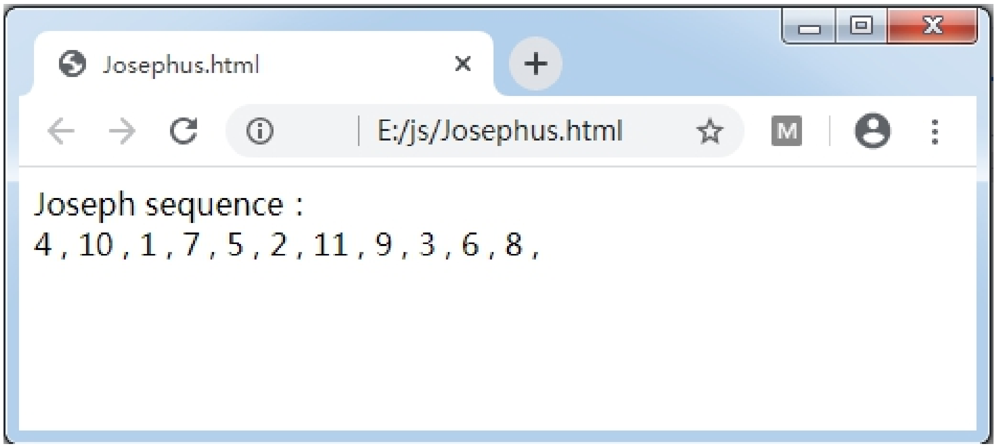

1
7
10
5
Start
4
10,11
2
8
11
6
3
9

**1. Create a Josephus.html with Notepad and open it in your browser.**

```
<script type ="text/javascript" >
    var N = 11 ;
    var M = 3 ;

    function joseph( man) {
        var count = 1 ;
        var i = 0 , pos = - 1 ;
        var alive = 0 ;
        while ( count <= N) {
            do {
                pos = ( pos + 1 ) % N; // Ring
                if ( man[ pos] == 0 )
                    i++;
                if ( i == M) {
                    i = 0 ;
                    break ;
                }
            } while ( true );
            man[ pos] = count;
            count++;
        }

    }

    ////////////////////testing//////////////////
    var man = new Array();
    for ( var i = 0 ; i < N; i++)
        man[ i] = 0 ;

    joseph( man);

   document.write( "\nJoseph sequenc e ：  <br>" );
    for ( var i = 0 ; i < N; i++)
        document.write( man[ i] + " , " );
```

```
</script>
```

## Result:


Josephus.html
E:/js/Josephus.html

```
Joseph sequence :
4 , 10 , 1 , 7 , 5 , 2 , 11 , 9 , 3 , 6 , 8 ,
```